

# 1<sup>st</sup> Year B.Sc (CS&IT)

## CT1114. Web Development

### *HTML, CSS, JavaScript*

## Section 1:

### Introduction and Course Overview

Dr. Sam Redfern

[sam.redfern@nuigalway.ie](mailto:sam.redfern@nuigalway.ie)

@psychicsoftware



NUI Galway  
OÉ Gaillimh

# Workshop Times

## Workshops:

1st Semester: Thursdays, 11am-1pm,  
Finnegan Suite (Top Floor, Block E, room BLE2012)  
2<sup>nd</sup> Semester: TBA

## Independent Study

In addition to the lecture notes and workshop exercises, you are expected to independently study each topic. Lecture notes alone do not provide sufficient depth.

## Assignments

Approximately every 2 weeks there will be an assignment to submit.  
Total marks for assignments: 20%.  
Total marks for written exam (in April/May): 80%.



# Overview of Syllabus

## Web Page Design

HTML: (HyperText Markup Language)

CSS: (Cascading Style Sheets)

## Client Side Scripting

JavaScript (JS) and Dynamic HTML (DHTML)



# Software

- You need a text (code) editor for editing code
- You need a web browser to run your work on

Good code editor software highlights code with colour coding of keywords, names, etc.

Good code editor software understands the programming language and will provide “intellisense” hints

## Recommended:

- Visual Studio Code. <https://code.visualstudio.com/>
- Or Sublime Text. <https://www.sublimetext.com/>
- Or notepad++. <https://notepad-plus-plus.org/downloads/>



# Web Browser (Web Client)



Firefox



Chrome

Powerful software which displays text, images, animations, etc. in a (mostly) consistent way, and supports user interactivity as well as interactions with web servers => quite capable of running full applications within a browser





# Discord



During (and outside) class, we will be using a Discord chatroom to share links (URLs) and so you can post comments and questions at any time

[\*\*https://discord.gg/NgXKRw3tdR\*\*](https://discord.gg/NgXKRw3tdR)

Please join the Discord Channel indicated in this link. (It's recommended that you sign up for a Discord account rather than appear using a different anonymous name every week).

Say 'Hi' in there.



NUI Galway  
OÉ Gaillimh

# Qwickly Code (Attendance Monitoring)

Use Blackboard  
Sidebar "Attendance" link (in the Assessment section)

To add Qwickly tool:

<https://tips.nuigalway.ie/m/88122/l/1090525-qwickly-attendance-for-instructors>



# HTML

HyperText Markup Language

The language used to define the contents of web pages  
Includes text, tables, hyperlinks, images, etc.

HTML is the language used to define webpage CONTENT and STRUCTURE

It is normally downloaded by your browser from a web server, and then interpreted and displayed (rendered) by your browser



# CSS

---

## Cascading Style Sheets

The language used to define the style of web pages  
Includes colours, fonts, sizes, positions, etc.

CSS is the language used to define webpage **VISUAL  
STYLE**

It is normally downloaded by your browser from a web server, and then interpreted by your browser and used to define the visual style of an HTML page that references it



# Javascript

---

A programming language that runs in the user's browser, providing interactivity, animation, calculations, and also the ability to change the HTML content of a page after it has been downloaded

Javascript is the language used to define the BEHAVIOUR and INTERACTIVITY of a webpage

It is normally downloaded by your browser from a web server, and then portions of it are executed as appropriate on an HTML page that references it



# W3Schools <http://www.w3schools.com>

Online tutorial site for internet languages

HTML, CSS, ASP, PHP, JavaScript, etc

“Try it Yourself” editor

A very good source of reference for your own independent study

Edit and Click Me >>

```
<html>
<body>

<h1>My First Heading</h1>

<p>My first paragraph.</p>

</body>
</html>
```

Your Result:

**My First Heading**

My first paragraph.

Edit the code above and click to see the result.



# A Simple HTML Page

---

```
<!DOCTYPE html>
<html>
  <head>
    <title>Simple Page</title>
  </head>
  <body>
    Hello World!
  </body>
</html>
```

This is a simplest-possible web page, without any CSS or Javascript. The `<head>..</head>` and `<body>..</body>` tags denote important sections in the document



# Another HTML Page, including paragraphs and spans

```
<!DOCTYPE html>
<html>
  <head>
    <title>Another simple page</title>
  </head>
  <body>
    <p>
      This is a paragraph.
    </p>
    <p>
```

This is another paragraph, containing a 'span' section that we could use Javascript to change the style or content of.

```
    </p>
  </body>
</html>
```



# Creating an HTML File

Open a Text Editor

e.g. Notepad++ or Visual Studio Code (VS Code)

**NB:** Microsoft Word is not a text editor!

Write some HTML

Save it as a .html file

e.g. mypage.html

Double click the file

Your page will open in your browser

*It has not been delivered from a webserver in the normal way*

However your browser is still happy to open it



# Exercise

Write a simple HTML page that contains at least two different paragraphs, as well as correct sections for head and body.



# Providing IDs for your tags

Wherever you use a tag in HTML, your browser will create a separate software object which has style and content.

By giving an ID to a tag, you will make it easy to refer to that specific software object using Javascript code, so that for example you can change its content or style using Javascript.



# Buttons

To create a clickable button in your page:

```
<button>Click Me!</button>
```

Click Me!

This should be inside the `<body>` section of the html document



# A page with a button that reacts to clicks by alerting the user with a message

```
<!DOCTYPE html>
<html>
<head>
  <title>A page with a button</title>
  <script>
    function handleClick() {
      alert("You clicked the button.");
    }
  </script>
</head>
<body>
  <button onclick='handleClick();'>Click Me!</button>
</body>
</html>
```



# A page with a button that reacts to clicks by modifying the content of a paragraph

```
<!DOCTYPE html>
<html>
  <head>
    <title>A page with a button</title>
    <script>
      function handleClick() {
        var p = document.getElementById('myParagraph');
        p.innerHTML = "..A stately pleasure dome decree.";
      }
    </script>
  </head>
  <body>
    <p id='myParagraph'>
      In Xanadu did Kubla Khan..
    </p>
    <button onclick='handleClick();'>Click Me!</button>
  </body>
</html>
```



# To see error messages, use..

In Firefox, use Tools > Web Developer > Browser Console

In Chrome, use More Tools > Developer Tools > (Console section)

For debugging in Javascript, you can also write directly to the console using  
**console.log("message");**



# Exercise

---

Make a webpage which contains a paragraph (containing some text) and two buttons.

When the first button is clicked, the content (innerHTML) of the paragraph should be changed.

When the second button is clicked, the content of the paragraph should be changed to something different.



# Exercise: sample solution

This sample solution will also indicate, in the paragraph, how many times each button has been clicked.

It will use variables to store these click counts.



# **CT1114**

# **Web Development**

# **HTML, CSS, JavaScript**

## **Section 2:**

## **Javascript Variables**

---

Dr. Sam Redfern

[sam.redfern@nuigalway.ie](mailto:sam.redfern@nuigalway.ie)

@psychicsoftware



NUI Galway  
OÉ Gaillimh

# Discord / Qwickly



Use our Discord channel for questions, comments etc. at any time

<https://discord.gg/NgXKRw3tdR>

**Qwickly Code (Attendance Monitoring)**



NUI Galway  
OÉ Gaillimh

# HTML

---

HyperText Markup Language

The language used to define the contents of web pages  
Includes text, tables, hyperlinks, images, etc.

HTML is the language used to define webpage CONTENT  
and STRUCTURE

It is normally downloaded by your browser from a web server, and then interpreted and displayed (rendered) by your browser



# Javascript

---

A programming language that runs in the user's browser, providing interactivity, animation, calculations, and also the ability to change the HTML content of a page after it has been downloaded

Javascript is the language used to define the BEHAVIOUR and INTERACTIVITY of a webpage

It is normally downloaded by your browser from a web server, and then portions of it are executed as appropriate on an HTML page that references it



# Reminder: structure of an HTML page with Javascript

```
<!DOCTYPE html>
<html>
  <head>
    <title>A page with a button</title>
    <script>
      function handleClick() {
        alert("You clicked the button.");
      }
    </script>
  </head>
  <body>
    <button onclick='handleClick()'>Click Me!</button>
  </body>
</html>
```



# What JavaScript is

- Scripting language
  - Client side in web-browser
- Consists of lines of executable code
  - Which are (typically) 'just in time' compiled
- Mostly located inside functions
  - Which may run in response to an event (e.g. onclick)
- Designed to add interactivity to HTML pages



# What JavaScript isn't

- JavaScript is **not** Java
- Java is a more **fully featured** programming language
  - More complex
  - Used for standalone applications, web services, etc.
  - Not sandboxed by the user's browser
- But Javascript is becoming more and more important and powerful in recent years (and on modern browsers)
- Also when run inside special executables:
  - Server-side programming (e.g. node.js)
  - Stand-alone applications (e.g. nodewebkit, cordova), web services (node.js) etc.



# What can JavaScript do?

- Give HTML designers a programming tool
- Can read/write Form contents (text boxes, drop-down lists etc.)
- Can handle (respond to) Events
  - onChange, onMouseOver, onLoad, etc
- Can read/write any attributes of HTML elements (e.g. innerHTML).
- Can even create and destroy elements on the page
  - Animation/games
  - Highly-interactive, responsive, animated user interfaces
  - Drag-drop
  - Etc.



# Where to put JavaScript code

- Inside HTML Head
  - Between <script ...> ... </script> tags
  - JavaScript code in functions in the head section will be executed only when CALLED
- Inside HTML Body
  - JavaScript code in the body section (not in functions) will be executed WHILE the page loads (use sparingly)
- External source files – for re-usable code
  - <script src="filename.js"></script>



# JavaScript Syntax

- 'C like' language
  - Like PHP, C++, Java, and others..
- Braces used to denote code blocks { }
- Semi-colons used at the end of lines
- Comments as per C
  - Single line comments // Single line
  - Block comment /\* Block comments  
can span multiple lines \*/



# Variables

- Created with the **var** keyword
  - You can also use **let** and **const** (recently added to the language)
- Syntax
  - `var name=value;`
  - `var num1, num2, num3;`
- Like most "scripting" languages, variables in Javascript do **not** need to be declared before being assigned a value
  - `var num=1;`
  - has the same effect as
  - `num=1;`
  - Variable "num" will be created if it doesn't exist
  - "`use strict";` is a good idea



# Variables

- Variables are used to store data values
- Most commonly, these are either numbers, or strings of text
- They can also be other things such as objects or functions (as we will see later)



# Arithmetic in JavaScript

- The same as C / C++ / Java etc
- Basic Arithmetic Operators

Addition	+	ans=a+7
Subtraction	-	ans=x-y
Multiplication	*	ans=a*b
Division	/	ans=a/b
Modulus	%	ans=x%y



# Exercise

```
<!DOCTYPE html>
<html>
  <head>
  </head>
  <body>
    <p id='myParagraph'>
      Result will display here
    </p>
    <button>Click for result!</button>
  </body>
</html>
```

- Modify this HTML page so that, when the button is clicked, the paragraph is modified so that it displays the result of the calculation:  $5+7*6$



# Parenthesis

- Parenthesis ( round brackets ) can be used to ensure order of priority in a computation
- $x = \underline{a+b+c}$       algebraically  
5
- $x=(a+b+c)/5$       algorithmically



# Order of Operations

- Set of rules, decide which operation is performed first
1. () innermost first
  2. \* / % left to right
  3. + - left to right



# Order of Operations

- Assuming  $a=4$ ,  $b=2$ ,  $c=5$  and  $d=3$ , evaluate the following, bearing the order of precedence in mind
  - $(a+b)*(c+d) =$
  - $a+b*c+d =$
  - $(d*(a-b))+c =$
  - $d*a-b+c =$



# Graded Exercise #1

- Make a web page which contains just a button
- When the button is clicked, a javascript function should be run
- The function should create four variables, named a, b, c, and d, with values as shown on the previous slide
- The function should then carry out the calculations shown on the previous slide, displaying each result using alert()



# **CT1114**

# **Web Development**

## **HTML, CSS, JavaScript**

### **Section 3:**

### **Expressions, Operators, and Loops in**

### **Javascript**

Dr. Sam Redfern

[sam.redfern@nuigalway.ie](mailto:sam.redfern@nuigalway.ie)

@psychicsoftware



NUI Galway  
OÉ Gaillimh

# Discord / Qwickly



Use our Discord channel for questions, comments etc. at any time

<https://discord.gg/NgXKRw3tdR>

**Qwickly Code (Attendance Monitoring)**



NUI Galway  
OÉ Gaillimh

# Expressions

- In programming, an expression is a statement that evaluates to a value
- The resulting value could for example be a number, a string, or a Boolean (true/false)

1 + 6

"Harry" + " " + "Potter"

20 > 19

var x = y \* z;

alert("My name is "+ "Harry" + " " + "Potter");



# Boolean Variables & Expressions

- A Boolean value may be either **true** or **false**
  - `var a = true;`
  - `var a = 20 > 19;`
- A Boolean expression is one that resolves to either true or false
  - Often, produced by the use of **relational** and **logical** operators (see below)



# Relational Operators

- Less than  $<$
- Greater than  $>$
- Less than or equal to  $\leq$
- Greater than or equal to  $\geq$
- Equal to  $=$
- Not equal to  $\neq$
- These operators all take two values (usually numbers) and produce a Boolean result by comparing them



# Conditional Statements

- **if** statement – execute code only if some condition is true
- **if...else** statement – execute some code if the statement is true and another piece if it is false
- **if...else if ... else** statements – used to execute one of many blocks of code
- **switch.. case** statement – used to execute one of many blocks of code
- Use braces (curly brackets) to denote blocks of code



# Example

```
<!DOCTYPE html>
<html>
  <head>
    <title>Inequality Demo</title>
    <script>
      function handleClick() {
        // create 2 random numbers
        var a = Math.floor( Math.random()*1000 );
        var b = Math.floor( Math.random()*1000 );

        // build up an output result string from some equality tests
        var res = "a = "+a+"  
b = "+b;
        if (a<b)
          res += "  
a is less than b";
        else
          res += "  
a is not less than b";

        if (a>b)
          res += "  
a is greater than b";
        else
          res += "  
a is not greater than b";

        // display the result
        var p = document.getElementById('myParagraph');
        p.innerHTML = res;
      }
    </script>
  </head>
  <body>
    <p id='myParagraph'>
      Results will display here
    </p>
    <button onclick='handleClick();'>Demo</button>
  </body>
</html>
```



# Exercise

- Edit the code from the previous slide so that it also indicates if a and b are equal



# Logical Operators

- `&&`
  - Logical 'and'
  - Compares two Boolean values and equates to true if they are both true
- `||`
  - Logical 'or'
  - Compares two Boolean values and equates to true if one or the other (or both) is true
- `!`
  - Logical 'not'
  - Negates the value of a Boolean value



# Example

```
<!DOCTYPE html>
<html>
  <head>
    <title>Three random numbers</title>
    <script>
      function handleClick() {
        // create 3 random numbers
        var a = Math.floor( Math.random()*1000 );
        var b = Math.floor( Math.random()*1000 );
        var c = Math.floor( Math.random()*1000 );

        // find the biggest
        var res = "a = "+a+"  
b = "+b+"  
c = "+c;
        if (a>b && a>c)
          res += "  
The biggest value is "+a;
        else if (b>a && b>c)
          res += "  
The biggest value is "+b;
        else
          res += "  
The biggest value is "+c;

        // display the result
        var p = document.getElementById('myParagraph');
        p.innerHTML = res;
      }
    </script>
  </head>
  <body>
    <p id='myParagraph'>
      Results will display here
    </p>
    <button onclick='handleClick();'>Demo</button>
  </body>
</html>
```



# Loops in JavaScript

- Repeat { block of code }
  - As long as some condition is true

## 'for' loops

- execute code a specific number of times

## 'while' loops

- execute code an undetermined number of times



# For Loop

- The For Loop repeats a block of code a certain number of times

```
for(var i=1; i<=10; i=i+1) {  
    // Execute this code on each loop iteration  
}
```

Three inner statements:

- for ( Initialisation; Test; Loop statement )



# Example

```
<!DOCTYPE html>
<html>
  <head>
    <title>Twenty random numbers</title>
    <script>
      function handleClick() {
        // create 20 random numbers and find the biggest
        var biggest = 0;
        for (var i=0; i<20; i++) {
          var num = Math.floor( Math.random()*1000 );
          if (num>biggest)
            biggest = num;
        }

        var res = "Of 20 random numbers, the biggest was
        "+biggest;

        // display the result
        var p = document.getElementById('myParagraph');
        p.innerHTML = res;
      }
    </script>
  </head>
  <body>
    <p id='myParagraph'>
      Results will display here
    </p>
    <button onclick='handleClick();'>Demo</button>
  </body>
</html>
```



# While Loop

- Executes code an undetermined number of times
- Loops while the condition remains true
  - Which could be zero times

```
var i = 0, j = 10000;  
  
while ( i < j ) {  
    // Execute this code per loop iteration  
  
    i++;  
  
    j /= 1.4;  
  
}
```



# Exercise

- How many times will this loop iterate?
- Write a webpage to find out!
- Use alert() to show your answer

```
var i = 0, j = 10000;  
  
while ( i < j ) {  
  
    // Execute this code per loop iteration  
  
    i++;  
  
    j /= 1.4;  
  
}
```



# Exercise (not graded – no need to submit)

- Edit the "twenty random numbers" example so that it shows the biggest, smallest, and sum of the numbers
- Bonus! - also display each of the 20 numbers



# **CT1114**

# **Web Development**

## **Section 4:**

## **Various HTML Tags including Spans, Hyperlinks and Images**

Dr. Sam Redfern

[sam.redfern@nuigalway.ie](mailto:sam.redfern@nuigalway.ie)

@psychicsoftware



NUI Galway  
OÉ Gaillimh

# Last Week's Final Exercise

- How many times will this loop iterate?
- Write a webpage to find out!
- Use alert() to show your answer

```
var i = 0, j = 10000;  
  
while ( i < j ) {  
  
    // Execute this code per loop iteration  
  
    i++;  
  
    j /= 1.4;  
  
}
```



# HTML Tags

- Tags denote markup **elements**
- Each tag is surrounded by angle brackets < >
- Tags normally (but not always) come in pairs:
  - the **opening tag** and the **closing tag**
- Tags are **not** case sensitive
  - <html> and <HTML> are functionally the same
  - Recommended to use lowercase (see good practice)
- Text between the tags is the **element content** or **inner HTML**
- Javascript can modify the innerHTML of an element (and its various other attributes)



# HTML Tags have Attributes

- HTML Tags have associated attributes (also called properties) that provide extra information to the browser
- Attributes consist of **name="value"** pairs
- Attributes are always added to the **opening tag**
- For example:
  - ***bgcolor*** is an attribute that **body** elements have
    - <body bgcolor="red">
- Example with two attributes defined:
  - <body bgcolor='red' onload='alert("loaded!");'>
  - Note the use of doublequotes inside singlequotes to avoid ambiguity..



# Some useful tags

Paragraph

<p>Text</p>

Line Break

<br>

Forces a line break wherever it's placed

Putting a carriage-return into the HTML code will *not* produce a visible line break!

Horizontal Rule

<hr>

HTML Comments

<!-- Text here --> **No ! at end**



# Headings

---

```
<h1> ... </h1>  
<h2> ... </h2>  
<h3> ... </h3>  
<h4> ... </h4>  
<h5> ... </h5>  
<h6> ... </h6>
```

Section 1  
Section 1.1  
Section 1.1.1  
Section 1.1.1.1  
Section 1.1.1.1.1



# HTML Lists

## Unordered Lists

Each List Item appears as Bullet

`<ul> </ul>`

`<li> </li>`

## Ordered Lists

Bullets replaced with numbers or letters  
Type of order defined with `type` attribute

`<ol> </ol>`



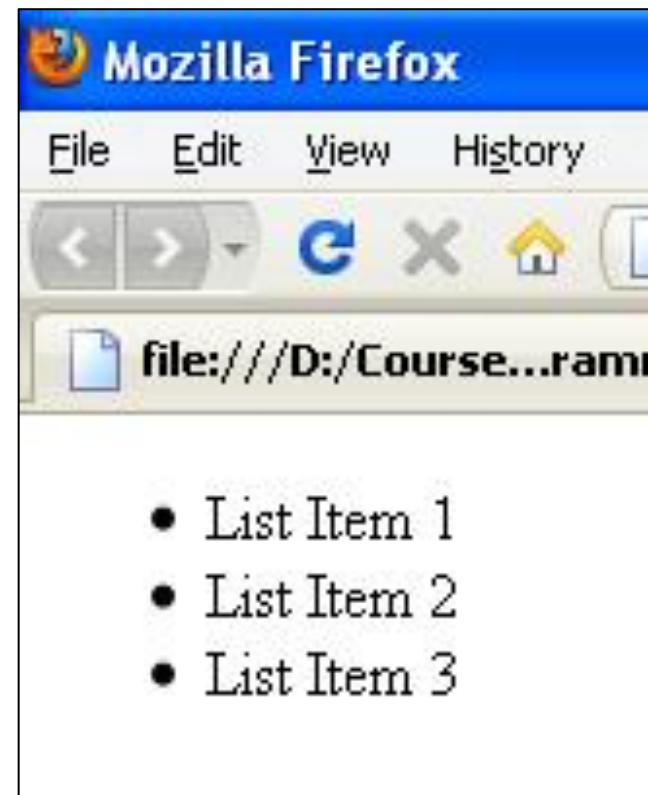
# Unordered Lists <ul>

```
<ul>
  <li>List Item 1</li>
  <li>List Item 2</li>
  <li>List Item 3</li>
</ul>
```

Bullet is default type

Other types can be specified

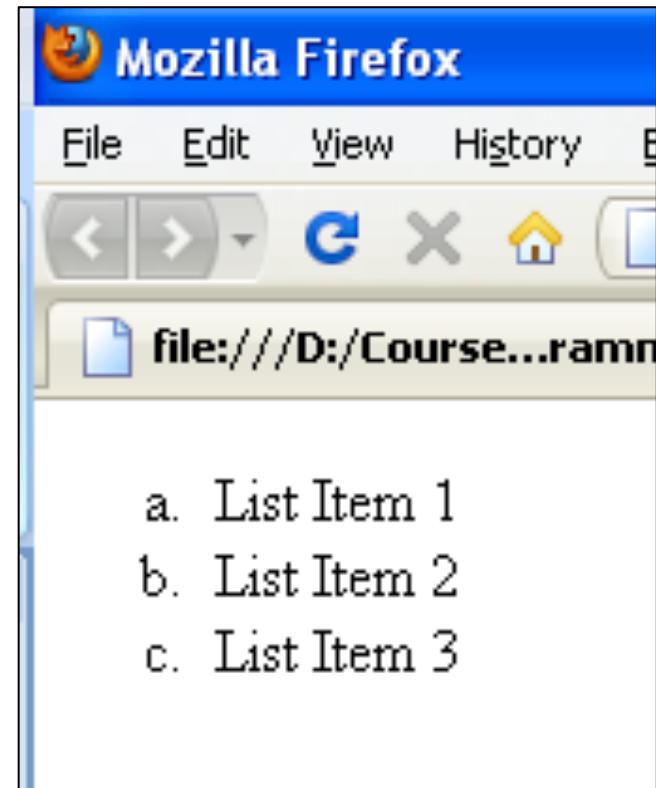
"disc", "square", "circle", etc



# Ordered Lists <ol>

```
<ol type="a">
  <li>List Item 1</li>
  <li>List Item 2</li>
  <li>List Item 3</li>
</ol>
```

Numeric is default type  
Other types can be specified  
"1", "A", "a", "I", "i"



# <span> .. </span>

```
<span> some content here </span>
```

spans let you define sections of content without line breaking. Useful for applying style to something, e.g.:

```
Some normal text and some <span  
style='color:red;'>red</span> text
```

style is a very important attribute that HTML Tags have, allowing you to define many visual settings for the Tag

Spans are useful for reading/writing innerHTML of a section of your page's text, using Javascript (see next slide)



# Example

```
<!DOCTYPE html>
<html>
<head>
    <title>Span click test</title>
    <script>
        var timesChanged = 0;
        function changeTheSpan() {
            timesChanged++;
            var msg = "has been changed "+timesChanged+" time";
            if (timesChanged>1)
                msg += "s";
            document.getElementById('mySpan').innerHTML = msg;
        }
    </script>
</head>
<body>
    This document contains a span whose content <span
id='mySpan' style='color:red;' onclick='changeTheSpan();'>may
be changed by clicking it</span>
</body>
</html>
```

# Exercise

Use HTML tags such as `<h1>`, `<p>`, `<hr>`, etc. to produce the depicted web page

The screenshot shows a web browser window with the title bar "File Edit View History Bookmarks Tools Help". The address bar shows "file:///C:/Users/sam/Desktop/literature/The Rime of the Ancient Mariner.html". The page content is as follows:

**The Rime of the Ancient Mariner**

**By Samuel Taylor Coleridge**

---

It is an ancient Mariner,  
And he stoppeth one of three.  
'By thy long grey beard and glittering eye,  
Now wherefore stopp'st thou me?

The Bridegroom's doors are opened wide,  
And I am next of kin;  
The guests are met, the feast is set:  
May'st hear the merry din.'

---

- This is, of course, not the whole poem
- It's just two verses
- But I'd recommend reading the rest!



# Images in HTML.

- Images are added with the empty tag <img>
  - “Empty” => it has no closing tag or innerHTML
- <img> requires a **source (src) attribute** with the URL of the image, e.g.:  
``
- Image (and other) filenames are **case sensitive** on Linux and Mac servers
  - .jpg is not the same as .JPG
- Format can be .jpg, .png, or .gif



# Hyperlink Anchors

<a> ... </a>

The anchor element <a> ... </a> is used to make clickable hyperlinks

- Use href attribute to define content to link to
- If content is on a different website:

`<a href="http://www.amazon.com/">Visit Amazon</a>`

- If content is on the same website:

`<a href="page2.html">Here is page 2</a>`

- If content is on the same page:

`<a name="faq1">FAQ #1</a>`

...

`<a href="#faq1">Click here for FAQ #1</a>`



# Absolute and Relative References

URL: The path to a resource

Any URL path in HTML can be given as either via  
an **Absolute** or a **Relative** Reference

Both types are valid, interchangeable and useful  
in different situations

e.g. Anchor tags and Image tags can use both  
Reference Types



# Absolute References

- Gives the complete (globally unique) path to resource
- Uses full URL, including protocol (http://) and path
- Absolute links reference a single, static, location
- To display the image "testimage.jpg" from the "images" folder within "www.randomsite.com":

```

```



# Relative References

- Gives the path to the resource *relative to* the HTML document in which it is being referenced
- Excludes the full URL of the resource
- Only pages local to the resource can use it, e.g. if testimage.jpg is in the same folder as the HTML file that references it:  
``
- Or if the image is, relative to the HTML file, inside a sub-folder:  
``



# Graded Exercise

- Write three separate web pages, which link to each other using hyperlinks, e.g.:

This image shows three separate web browser windows, each displaying a different page (A, B, or C) with hyperlinks to the other two pages.

- Page A:** "This is page 'A'. You can visit below:  
[Page A](#)  
[Page B](#)  
[Page C](#)"
- Page B:** "This is page 'B'. You can visit below:  
[Page A](#)  
[Page B](#)  
[Page C](#)"
- Page C:** "This is page 'C'. You can visit all of our informative pages below:  
[Page A](#)  
[Page B](#)  
[Page C](#)"

The browser interface includes a menu bar (File, Edit, View, History, Bookmarks, Tools, Help), a tab bar (Google, Another simpl, Iteration test, Learning, The Rime of ti, /C:/Users/), and a toolbar with various icons.



# CT1114

# Web Development

HTML: Images

Javascript: `window.alert()`, `window.prompt()`

Dr. Sam Redfern

[sam.redfern@nuigalway.ie](mailto:sam.redfern@nuigalway.ie)

@psychicsoftware



NUI Galway  
OÉ Gaillimh

# Last Week's Graded Exercise

- Write three separate web pages, which link to each other using hyperlinks, e.g.:

This image shows three separate browser windows, each displaying a different page (A, B, or C) from a local file path. Each page contains links to the other two pages, illustrating a simple web structure.

**Page A (Left Browser Window):**

This is page 'A'. You can visit below:  
[Page A](#)  
[Page B](#)  
[Page C](#)

**Page B (Middle Browser Window):**

This is page 'B'. You can visit below:  
[Page A](#)  
[Page B](#)  
[Page C](#)

**Page C (Right Browser Window):**

This is page 'C'. You can visit all of our informative pages below:  
[Page A](#)  
[Page B](#)  
[Page C](#)

**Page Footer:**

NUI Galway  
OÉ Gaillimh

# Images in HTML.

- Images are added to a page with the empty tag `<img>`
- `<img>` requires a **source (src) attribute** with the URL of the image
- Image (and other) filenames are **case sensitive** on Linux and Mac servers
  - .jpg is not the same as .JPG
- The src can use either a **relative** or an **absolute** address,

e.g absolute:

```

```

e.g. relative:

```

```



# Exercise

- Download the referenced png file and store it in the correct place so that this HTML page, with its relative reference, displays both images correctly
- **Do not edit the HTML code itself**

```
<!DOCTYPE html>
<html>
  <head>
  </head>
  <body>
    <img src='http://www.psychicsoftware.com/ct1113/banner-image-1.png'><br>
    <img src='images/banner-image-1.png'>
  </body>
</html>
```



# The window Object

- Represents an open window in a browser
- **window** is an object (as is **document**)
- Objects have associated **attributes** (pieces of data) and **methods** (things they can do)
- Some important window methods:
  - alert
  - prompt
  - confirm
  - setTimeout



# The alert Method

- Displays a **Modal** dialog box containing a message
- Takes a string argument
- Syntax:

```
window.alert("Hello World!");
```

```
alert("Hello World!"); // this also works
```



# The prompt Method

- Prompt the user to enter input via a **Modal** dialog box

```
var result = window.prompt("string1", "string2");
```

*First argument is the prompt's text*

*Second argument is the default value in the field  
(optional)*



# The prompt Method

- Prompt has a return value

```
var name;
```

```
name = window.prompt("What is your name?",  
"Enter your name");
```



# Exercise

- Make a webpage which uses `prompt()` to ask the user for their name, and which then writes a greeting (to include their name) into a paragraph on the page:



# Graded Exercise

- Create a web page which displays an image (using absolute referencing) from the following URL:  
<http://www.psychicsoftware.com/ct1113/banner-image-1.png>
- When the image is clicked, you should (with Javascript) change its **src** attribute to:  
<http://www.psychicsoftware.com/ct1113/banner-image-2.png>
- On each subsequent click, display the next image (banner-image-3.png, banner-image-4.png, banner-image-5.png, banner-image-6.png)
- After image 6, revert to image 1
- Submit your code via Blackboard for grading



# CT1114

# Web Development

## HTML, CSS, JavaScript

### Javascript: window.confirm(), Parse functions, Debugging tools in Chrome and Firefox

---

Dr. Sam Redfern

[sam.redfern@nuigalway.ie](mailto:sam.redfern@nuigalway.ie)

@psychicsoftware



NUI Galway  
OÉ Gaillimh

# Last Week's Graded Exercise

- Create a web page which displays an image (using absolute referencing) from the following URL:  
<http://www.psychicsoftware.com/ct1113/banner-image-1.png>
- When the image is clicked, you should (with Javascript) change its **src** attribute to:  
<http://www.psychicsoftware.com/ct1113/banner-image-2.png>
- On each subsequent click, display the next image (banner-image-3.png, banner-image-4.png, banner-image-5.png, banner-image-6.png)
- After image 6, revert to image 1

# The Window Object

- Represents an open window in a browser
- **window** is an object just like **document**
- Objects have associated **attributes** (pieces of data) and **methods** (things they can do)
- Some important window methods:
  - alert (covered last week)
  - prompt (covered last week)
  - confirm (see below)
  - setTimeout (see later)

# The Confirm Method

- Used if you want the user to verify or accept something
- User presented with the choice of clicking **OK** or **Cancel** to proceed
- If the user pressed **OK**
  - **true** is returned
- If the user pressed **Cancel**
  - **false** is returned

# The Confirm Method

- Syntax:

```
var choice;
```

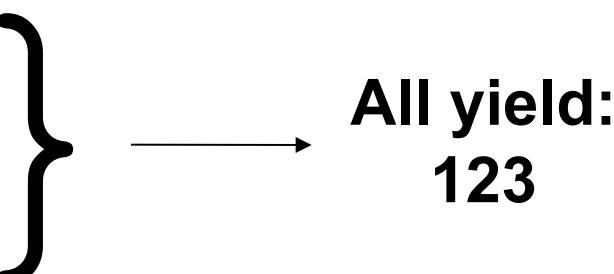
```
choice=confirm("Confirm Dialogs give you a choice!");
```



# Confirm Example

```
<!DOCTYPE html>
<html>
<head>
<script>
function giveChoice() {
    if (confirm("Make the document red?") == true)
        document.body.style.background = "red";
    else {
        alert("Right so, it'll be green.");
        document.body.style.background = "green";
    }
}
</script>
<head>
<body onload="giveChoice();">
</body>
</html>
```

# Parse Functions

- `parseInt(string)`
    - Converts the string to an Integer and returns it
  - `parseFloat(string)`
    - Converts the string to a Floating Point number
  - String argument must **not** be empty
  - String argument must **start** with a valid number
  - `parseInt("123");`
  - `parseInt("123abc");`
  - `parseInt("123abc456");`
- 
- All yield:  
123

# Parse Functions - isNaN() - "Is Not a Number"

```
var number = parseInt("abc");  
alert("The number is: " + number + " and  
multiplied by two is " + (number*2));
```

Output: *The number is NaN and multiplied by two is NaN*

- If string is not a number, it is set to a special **NaN** value
- **isNaN(value)** checks if a value is a number
- Returns **true** if value is Not a Number
- Returns **false** if value is a Number

# isNaN() function in use

```
<!DOCTYPE html>
<html>
  <head>
    <script>
      function test() {
        var str = prompt("Please type a number");
        if (isNaN(str)) {
          alert("That's not a number");
        }
        else {
          var number = parseInt(str);
          alert("The number is: " + number + " and multiplied by two
is " + (number*2));
        }
      }
    </script>
  </head>
  <body>
    <button onclick='test();'>Click Me!</button>
  </body>
</html>
```

# Exercise

- Make a webpage which, **as soon as it loads**, prompts the user for an integer between 1 and 100, using the `prompt()` method
- If the input given cannot convert to a number (e.g. if they type their name), or if it converts to a number that's not in the correct range, tell them the error. Otherwise, thank them.

# Debugging Tools

Firefox

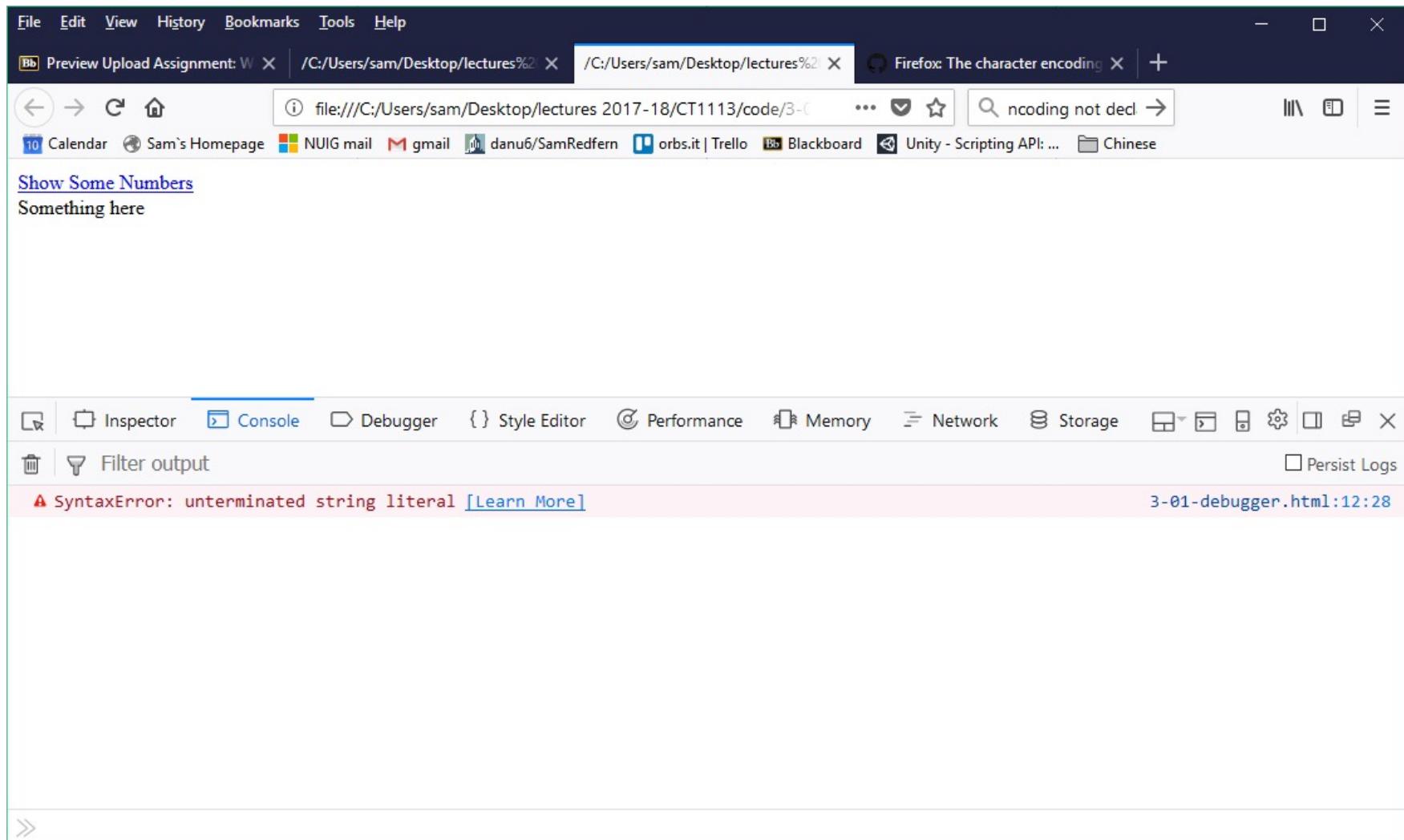
Chrome

# Consider this HTML+Javascript..

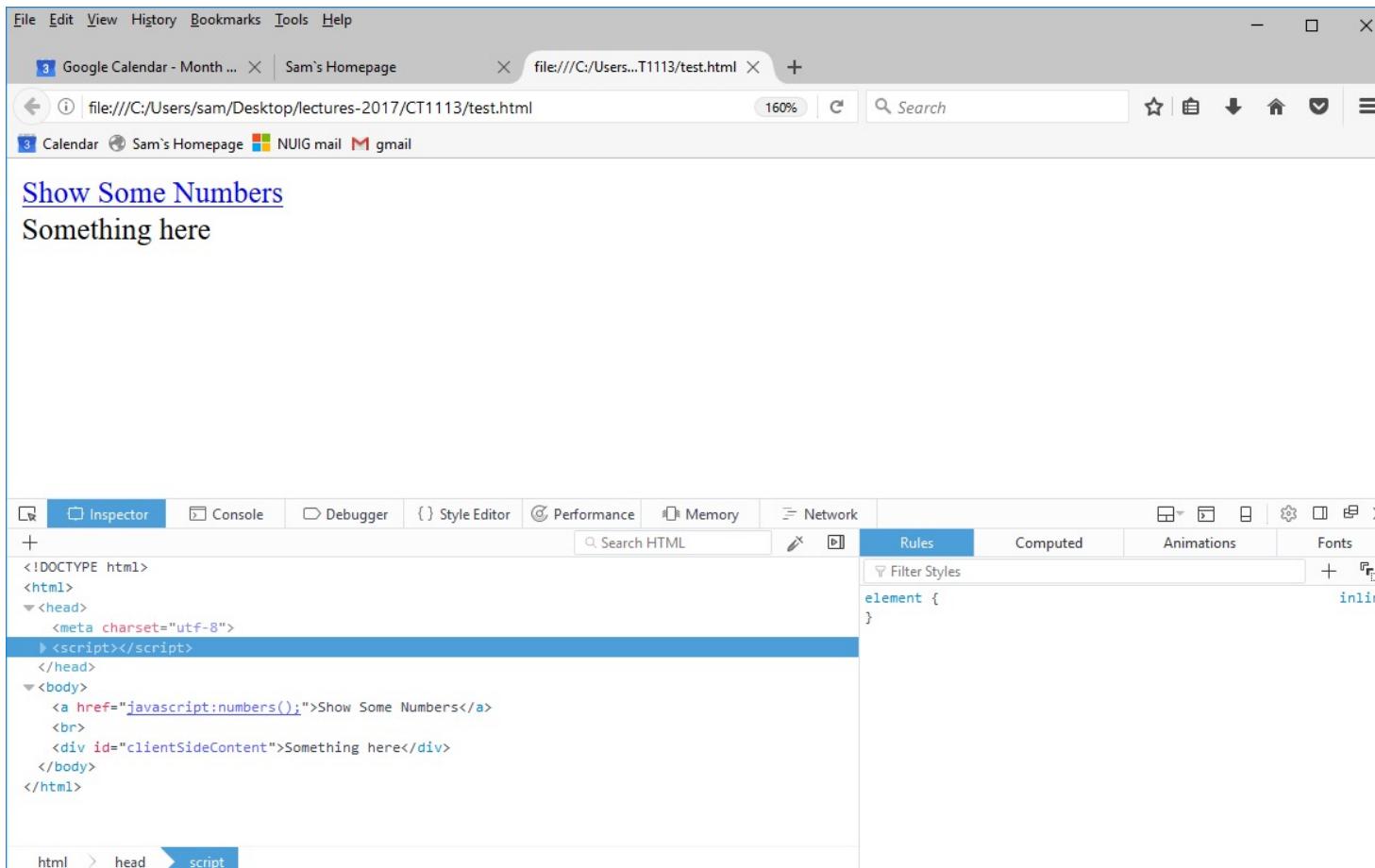
```
<!DOCTYPE html>
<html>
<meta charset="utf-8"/>
<head>
<script>
    function numbers() {
        var sHTML = "<p>";
        for (var i=1; i<=20; i++) {
            sHTML += Math.random() + "<br>";
        }
        sHTML += "</p>";
        document.getElementById("clientSideContent").innerHTML = sHTML;
        console.log("Final HTML generated in loop: "+sHTML);
    }
</script>
</head><body>
    <a href='javascript:numbers();'>Show Some Random Numbers</a><br>
    <div id='clientSideContent'>Something here</div>
</body>
</html>
```

**Does it work? If not, why not?  
The Web Console might help**

# Web Console (Firefox)

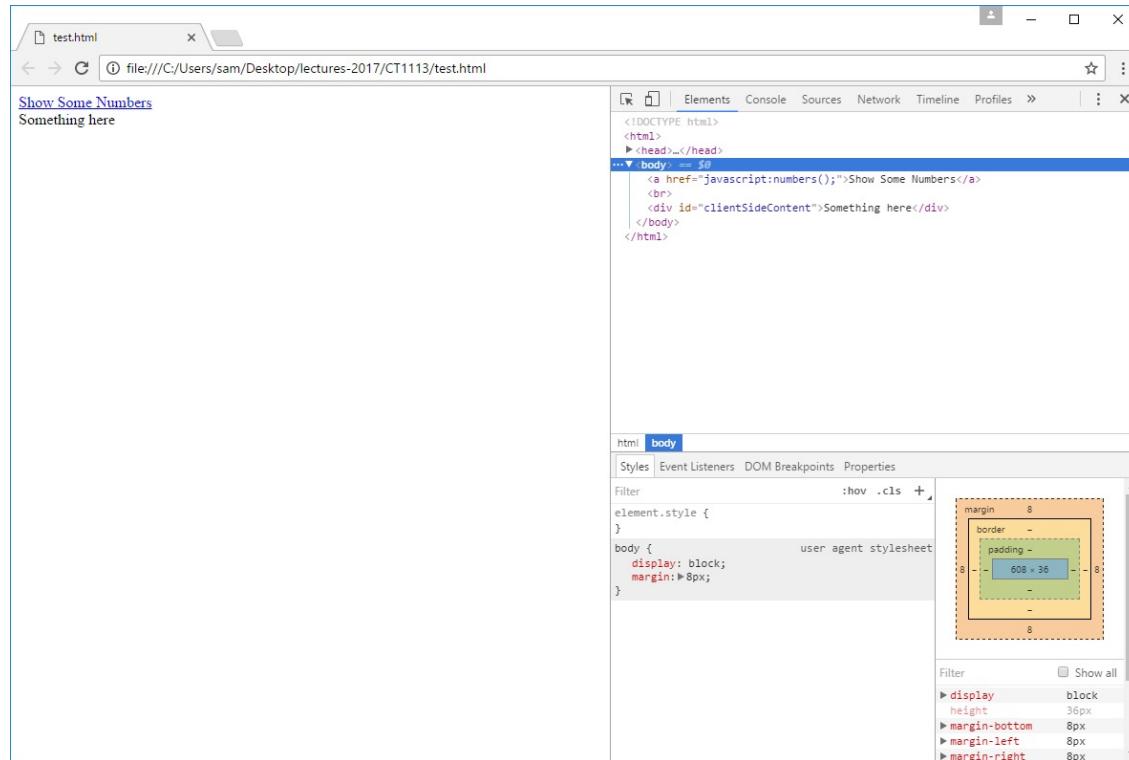


# Firefox: Inspector

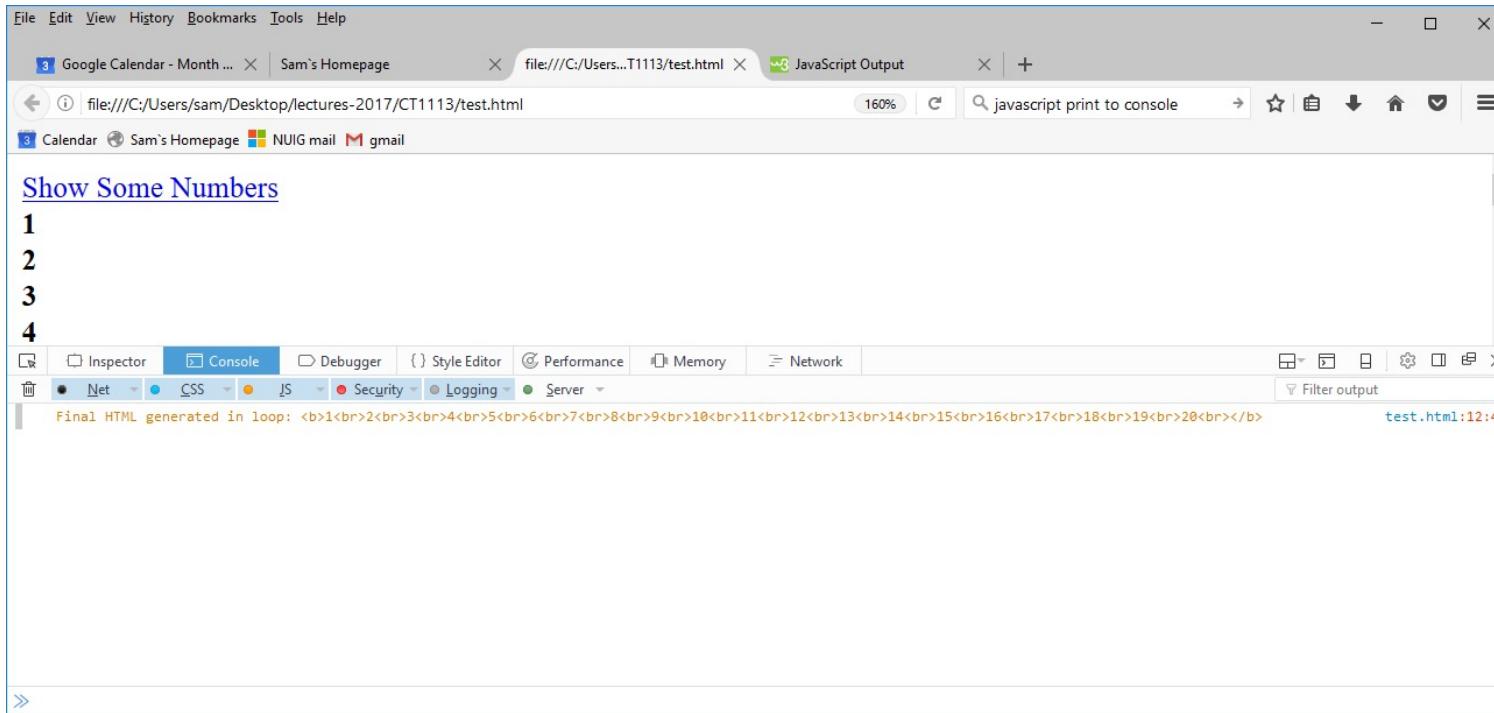


- Browse the live Document Object Model (DOM)
- Even make changes!

# Chrome: Elements = Firefox: Inspector

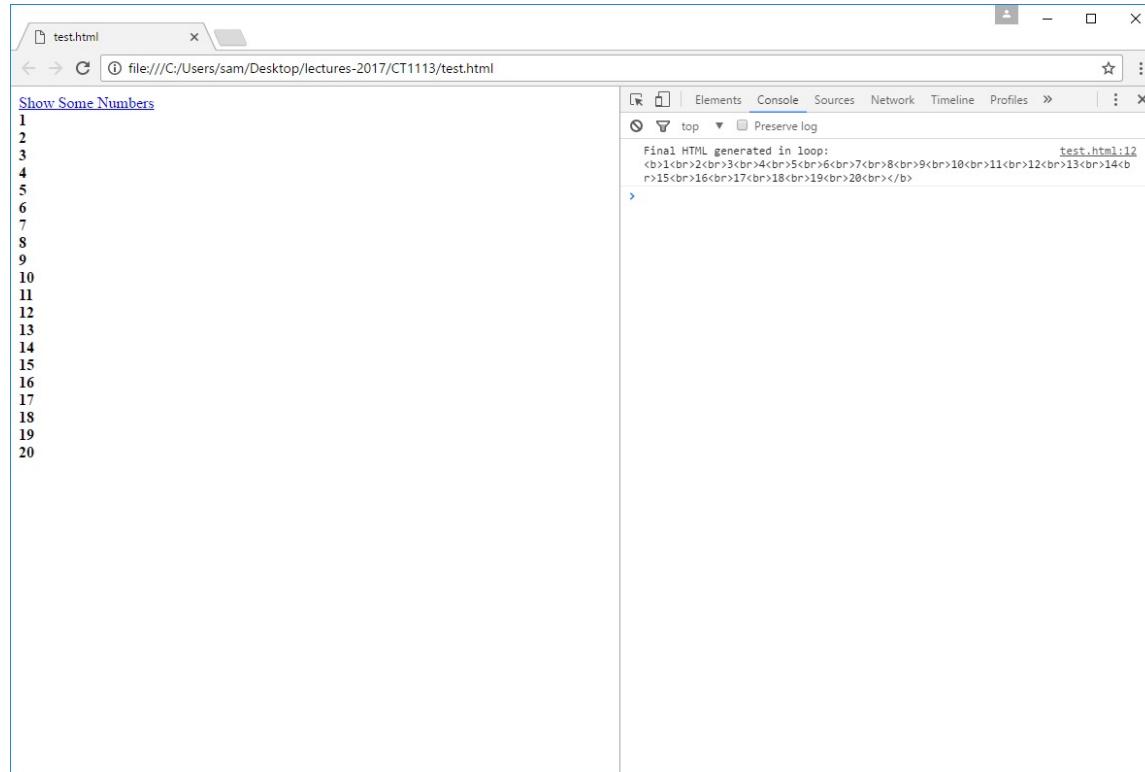


# Firefox: Web Console



- Use `console.log(value);` to write data to the web console, to see the sequence/flow-of-control and values of variables during execution

# Chrome: Console = Firefox: Web Console



# Firefox: Debugger

The screenshot shows the Firefox Developer Tools debugger open in a browser window. The tabs at the top are File, Edit, View, History, Bookmarks, Tools, Help, Google Calendar - Month ..., Sam's Homepage, file:///C:/Users/sam/Desktop/lectures-2017/CT1113/test.html, and JavaScript Output. The main content area displays a simple HTML page with a link to 'Show Some Numbers' and a div containing 'Something here'. The Firefox toolbar below the tabs includes Back, Forward, Stop, Reload, Home, and other icons.

The debugger interface has several tabs: Inspector, Console, Debugger (which is selected), Style Editor, Performance, Memory, and Network. The Sources tab is also selected. The code editor shows the following JavaScript:

```
javascript:numbers();
file:///C:/Users/sam/Desktop/lectures-2017/CT1113/test.html:1:1
<!DOCTYPE html>
<html><head>
<meta charset="utf-8"/>
<script>
function numbers() {
    var sHTML = "<b>";
    for (var i=1; i<=5; i++) {
        sHTML += i + "<br>";
    }
    sHTML += "</b>";
    clientSideContent.innerHTML = sHTML;
    console.log("Final HTM generated in loop: " + sHTML);
}
</script>
</head><body>
<a href="javascript:numbers();">Show Some Numbers</a><br>
<div id='clientSideContent'>Something here</div>
</body></html>
```

The Variables panel on the right shows the current state of variables:

- Function scope [numbers]:
  - this: Window → test.html
  - i: 5
  - sHTML: "<b>1<br>2<br>3<br>4<br>5<br>"
- Global scope [Window]:

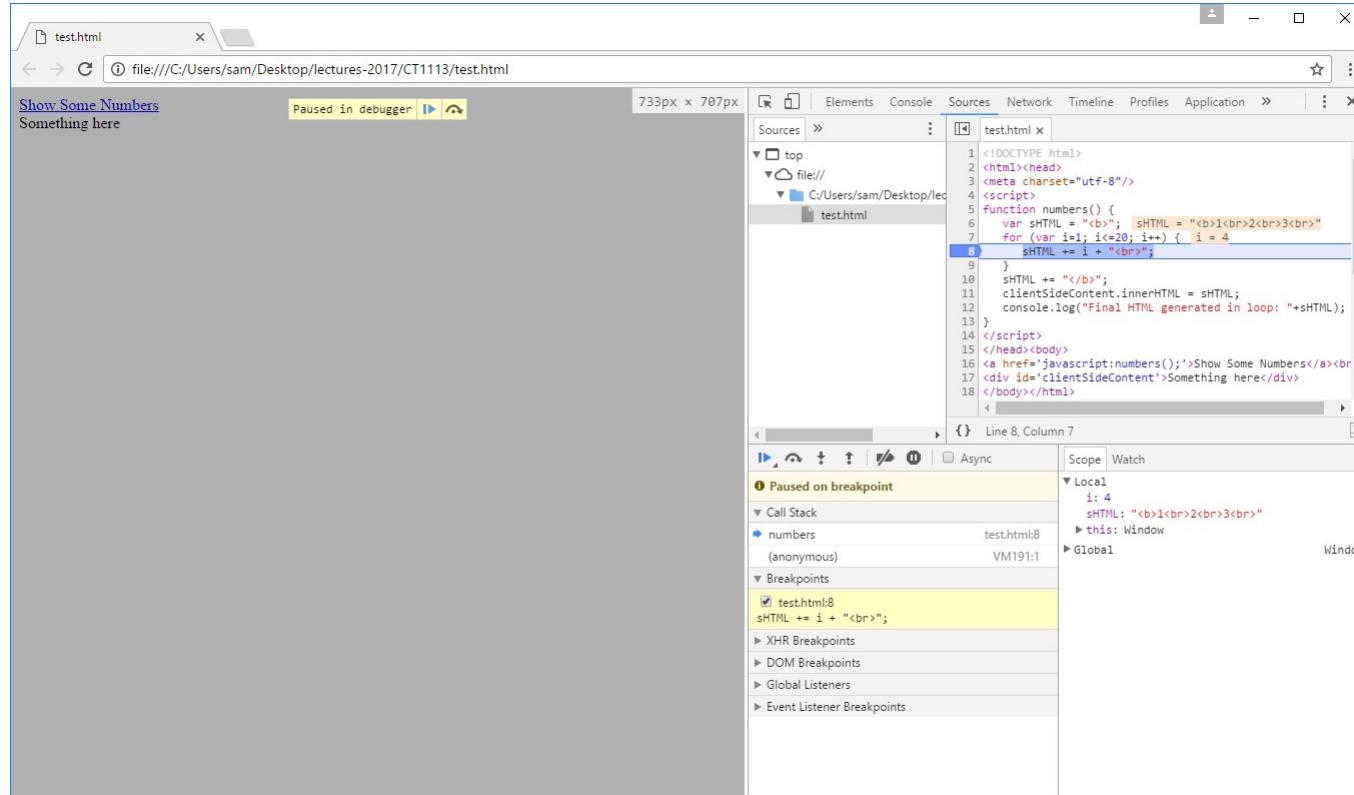
Annotations with blue arrows point to specific elements:

- 'Resume' points to the resume button in the toolbar.
- 'Step Over' points to the step over button in the toolbar.
- 'Step In' points to the step in button in the toolbar.
- 'Step Out' points to the step out button in the toolbar.
- 'Rightclick, Add Breakpoint (or leftclick in left margin)' points to the code editor's left margin where a breakpoint is set.
- 'Observe variables as you step line-by-line' points to the Variables panel.

Rightclick, Add Breakpoint  
(or leftclick in left margin)

Observe variables as you step line-by-line

# Chrome: Sources = Firefox: Debugger



# Debugging Exercise

```
<!DOCTYPE html>
<html>
  <head>
    <script>
      function isPrime(n) {
        for (var i=2; i<=n/2; i++) {
          if (n%i==0)
            return false;
        }
        return true;
      }

      function showPrimes() {
        var output = "";
        var num = 0;
        for (var i=2; i<5000; i++) {
          if (isPrime(i)) {
            num++;
            output += i;
            if (num%10==0)
              output += "<br>";
            else
              output += " &nbsp; ";
          }
        }
        document.getElementById("pOutput").innerHTML = output;
      }
    </script>
  </head>
<body onload="showPrimes();">
  <p>Here's some prime numbers:</p>
  <p id='pOutput'></p>
</body>
</html>
```

The Javascript on this webpage has 3 errors.

Fix this code so that prime numbers are displayed

# Exercise (not graded)

- Building on the previous exercise where you obtained a number between 1-100 from the user, and checked its validity
- If the input from the user does convert to a number in the correct range (1-100), produce output in the webpage which counts from 1 up to their number, with a line-break <br> between each number
- Hint: use a 'for' loop to build up your output string and assign it into the innerHTML of a <span> or <p>

# **CT1114**

# **Web Development**

# **HTML, CSS, JavaScript**

## **Section 7 part 1:**

## **HTML text fields, Javascript strings**

---

Dr. Sam Redfern

[sam.redfern@nuigalway.ie](mailto:sam.redfern@nuigalway.ie)

@psychicsoftware



NUI Galway  
OÉ Gaillimh

# HTML tag:

```
<input type='text'>
```

## Text Input Field

```
<input type="text" id="txtBox">
```

id attribute used to identify the field in code

## Specify default text with value attribute

```
<input type="text" value="Initial Text"  
id="txtBox">
```

## Can specify size through style attribute

```
<input type="text" style="width:50px;"  
id="txtBox">
```

# Exercise

- Create a web page which provides:
  - an <input> box for the user to enter a number into
  - a button which the user clicks
- When the button is clicked, the user should be shown, in a <div> tag, the factors of their number.
  - The .value property tells you what the user has entered in the box
- If you have spare time, include some error checking (as shown below)
- See starting code on next slide

64

Show Factors

The factors of 64 are: 1, 2, 4, 8, 16, 32, 64.

Hello

Show Factors

Please type a positive number. 'Hello' is not a positive number

```
<!DOCTYPE html>
<html>
<meta charset="utf-8"/>
<head>
<script>
    function showFactors() {
        }
</script>
</head><body>
    <input type='text' id='txtInputBox' value='1'>
    <button onclick="showFactors();">Show Factors</button>
    <div id='divOutput'>Output will go here</div>
</body>
</html>
```

**HTML tag:**

**<textarea> ... </textarea>**

Accepts multiple lines of text

Uses rows and cols attributes to define size

```
<textarea rows="20" cols="40"  
id="myTextArea" value="Default  
text goes here"></textarea>
```

# Strings

- A **string** is a group of characters
- A **string literal** is a group of characters enclosed in quotes
  - "This is a string literal"
  - 'This is too'
  - " "This" is not"
  - " 'This' is"
  - This isn't
- A **string variable** is a variable that holds a string
- Space is a valid character in a string (as are other special characters)

# Concatenating Strings

- String operator ‘+’ is used to join two strings

```
var name;
```

```
name = "John" + "Doe";
```

- We can of course also concatenate string variables

```
var first, last, full;
```

```
first = "John";
```

```
last = "Doe";
```

```
full = first + last;
```

```
var name = "John Smyth";
```

```
alert("Hello there, " + name + "!");
```

# Escape Sequences

- Special Characters in a string

\n              New line

\r              Carriage Return

\t              Tab

\\"              Backslash

\\"              Double Quote

\'              Single Quote

# Some string handling methods

These are methods of the string object. Hence the dot operator '.'.

```
var a = "something";
```

a.length - returns the number of characters in the string

a.indexOf("s") - returns the 1st position at which "s" occurs in string  
- position is a 0-based index

- also .indexOf("s",5) to start looking at position 5

a.substr(3,5) – returns a portion of the string, in this case we start at position 3 (i.e. the 4<sup>th</sup> character) and get 5 characters

Example that uses string handling. For code, see next slide:

Your Name: Sam Redfern

Process Name

Hi Sam Redfern, you have 11 characters in your name

The space character first appears at position 3

Your first name is: Sam

Your surname is: Redfern

# Example: string handling methods

```
<!DOCTYPE html>
<html>
  <head>
    <script>
      function processName() {
        var myname = document.getElementById("txtName").value;
        var len = myname.length;
        var output = "Hi "+myname+", you have "+len+" characters in your
name<br>";
        var spos = myname.indexOf(" ");
        output += "The space character first appears at position "+spos+"<br>";
        var firstname = myname.substr(0,spos);
        output += "Your first name is: "+firstname+"<br>";
        var surname = myname.substr(spos+1,len-spos-1);
        output += "Your surname is: "+surname+"<br>";
        document.getElementById("divOutput").innerHTML = output;
      }
    </script>
  </head>
<body>
  Your Name: <input type='text' id='txtName' value=' '><br>
  <button onclick="processName();">Process Name</button><br>
  <div id='divOutput'>Output will go here</div>
</body>
</html>
```

# Exercise (not graded)

- Create a web page which provides:
  - two  boxes for the user to enter numbers
  - Four  objects, labelled +, -, \*, /
  - A 

tag, for displaying output
- When a button is clicked, the result of using the chosen operator (add, subtract, multiply, or divide) on the two numbers should be displayed
- See starting code (next slide)

**The Little Calculator (TM)**

The screenshot shows a web-based calculator interface titled "The Little Calculator (TM)". It features two input fields at the top containing the numbers "250" and "102". Below these are four buttons labeled "+", "-", "\*", and "/". The "\*" button is highlighted with a blue border. At the bottom of the interface, the result of the multiplication is displayed as "250\*102=25500".

```
<!DOCTYPE html>
<html>
<meta charset="utf-8"/>
<head>
</head><body>
<h2>The Little Calculator (TM)</h2>
<input type='text' id='txtInputBox1'>
<input type='text' id='txtInputBox2'><br>
<button>+</button> <button>-</button>
<button>*</button> <button>/</button>
<div id='divOutput'></div>
</body>
</html>
```

# **CT1114**

# **Web Development**

# **HTML, CSS, JavaScript**

## **Section 7 (part 2):**

## **HTML text fields, Javascript strings**

---

Dr. Sam Redfern

[sam.redfern@nuigalway.ie](mailto:sam.redfern@nuigalway.ie)

@psychicsoftware



# Example:

As the user types into a <textarea>, their text is displayed one word per line

```
Pack my box with five dozen liquor jugs
```

Pack  
my  
box  
with  
five  
dozen  
liquor  
jugs

(code is on next slide)

```
<!DOCTYPE html>
<html>
  <head>
    <script>
      function updateOutput() {
        var txt = document.getElementById("txtUserInput").value; // text typed by
user
        var numchars = txt.length;
        var txtOutput = "";
        var currWord = ""; // initial word
        for (var i=0; i<numchars; i++) {
          var currLetter = txt.substr(i,1);
          if (currLetter==" ") { // space character => end current word and
start next one
            txtOutput += currWord+<br>;
            currWord = "";
          }
          else
            currWord += currLetter; // add letter to current word
        }
        if (currWord.length>0) // deal with any remaining word that didn't end
with a space
          txtOutput += currWord;
        document.getElementById("pOutput").innerHTML = txtOutput; // show result
to user
      }
    </script>
  </head>
  <body>
    <textarea id='txtUserInput' onkeyup="updateOutput();" value=""></textarea><br>
    <p id='pOutput'>
      Your output will be shown here.
    </p>
  </body>
</html>
```

# Exercise

- Edit the last example so that:
  1. Rather than displaying each word on a separate line, it displays each letter on a separate line
  2. It displays those letters in reverse order

# Exercise

- Edit the previous example so that:
  1. It counts and displays how many vowels appear in the text (a,e,i,o,u,A,E,I,O,U)
  2. It adds up and displays each numerical digit, e.g. if the text was "Hello 123 my name is Sam 456", then the sum of numerical digits would be  $1+2+3+4+5+6 = 21$

# Graded Exercise

- Create a web page which presents a <textarea> to the user, and as they type it constantly tells them how many characters and how many words they have typed.
- Hint: the 'onkeyup' method will happen for the <textarea> each time any key is pressed in it.
- Submit your code via Blackboard.

Volumetric Fog & Mist is the enhanced version of Dynamic Fog & Mist (which is also included in the package) and has been designed to provide a better looking and more flexible fog, including fog areas and cloud formations with support of lighting and glow effects.

264 characters  
46 words.

# How to make the word count more robust?

This sentence contains some multiple spacing,  
A carriage-return and some words with punctuation that's likely to  
cause problems for simple word count algorithms

This ·sentence ·· ·contains ·some ·· ·multiple ·spacing, ¶  
A ·carriage-return ·and ·some ·words ·with ·punctuation ·that's ·likely ·to ·  
cause ·problems ·for ·simple ·word ·count ·algorithms

## How many words?

- Counting spaces gives 26
- But really there's 24 words
- How could we better define what a word is, rather than  
pretending that a word is a space?

# CT1114

# Web Development

## Section 8:

## HTML Tables

Dr. Sam Redfern

[sam.redfern@nuigalway.ie](mailto:sam.redfern@nuigalway.ie)

@psychicsoftware



NUI Galway  
OÉ Gaillimh

# Tables in HTML

Arrange data in rows and columns of cells

Cells can contain any HTML data

Text, images, links, forms, form fields, other tables, etc.  
i.e. each cell has *innerHTML*

Cells can span multiple rows and columns

Via the **rowspan** and **colspan** attributes

Tables are created with the `<table>` tag

Tables are closed with the `</table>` tag

# Rows and Columns

Define a row in the table **table row** tags

Start the row with `<tr>`

End a row with `</tr>`

Each row can have multiple columns

Column headings created with the **table heading** tag

`<th>Heading</th>`

Cell data can be created with the **table data** tag

`<td>Cell data here</td>`

Each row of a table should have the same number of columns

# Table Example

```
<table border="1">
    <caption>A simple table</caption>
    <tr>
        <th>Column 1</th>
        <th>Column 2</th>
    </tr>
    <tr>
        <td>100</td>
        <td>200</td>
    </tr>
</table>
```

A simple table

Column 1	Column 2
100	200

# Spanning Multiple Columns

Cells can span multiple columns in a table

Use the **colspan** attribute

```
<tr>
    <th>Name</th>
    <th colspan="2"
```

Name	Telephone Number
Bill Gates	555 12345 555 99887

# Spanning Multiple Rows

Cells can span multiple rows in a table  
Use the **rowspan** attribute

```
<tr>
    <th>First Name</th>
    <td>Bill Gates</td>
</tr>
<tr>
    <th rowspan="2">Telephone</th>
    <td>555 77 854</td>
</tr>
<tr>
    <td>555 77 855</td>
</tr>
```

<b>First Name</b>	<b>Bill Gates</b>
<b>Telephone</b>	<b>555 77 854</b>
	<b>555 77 855</b>

# Formatting Tables

Borders can be added to tables by adding the "border" attribute to the table tag

```
<table border="1">
```

Cell padding sets the number of pixels surrounding the cell's contents

```
<table cellpadding="30">
```

Cell spacing sets the number of pixels between each cell wall

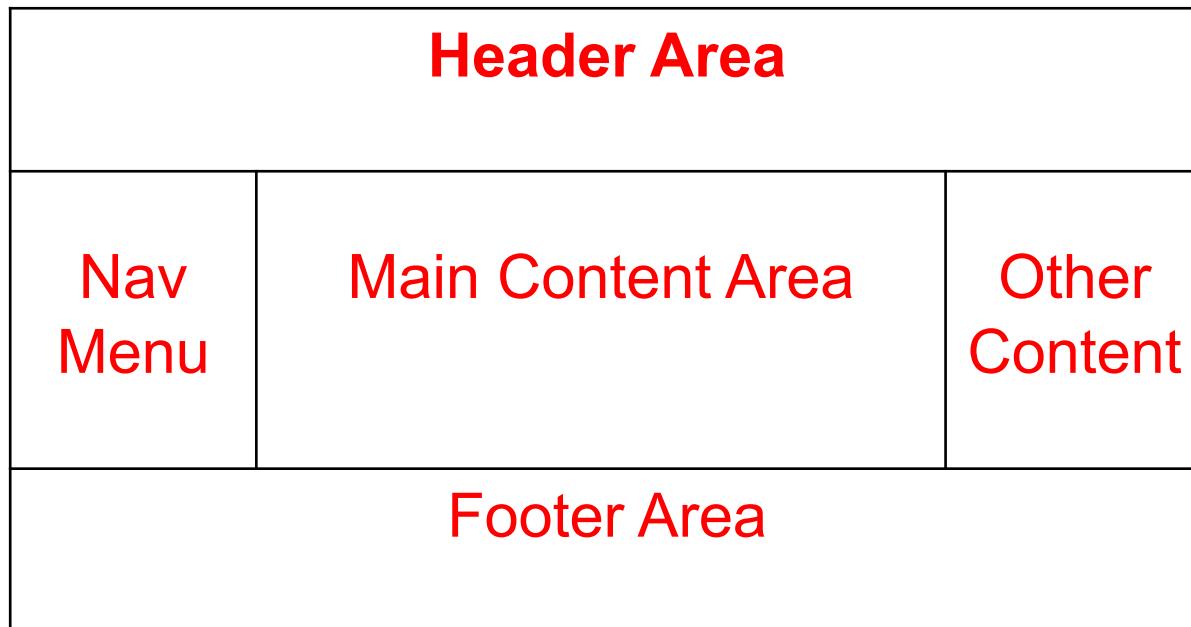
```
<table cellspacing="10">
```

Cell data can be aligned using the "align" attribute  
"center", "left", "right"

Or vertically with "valign" attribute which takes the values "top", "middle", "bottom"

# A Common Web Page Layout

A common arrangement:

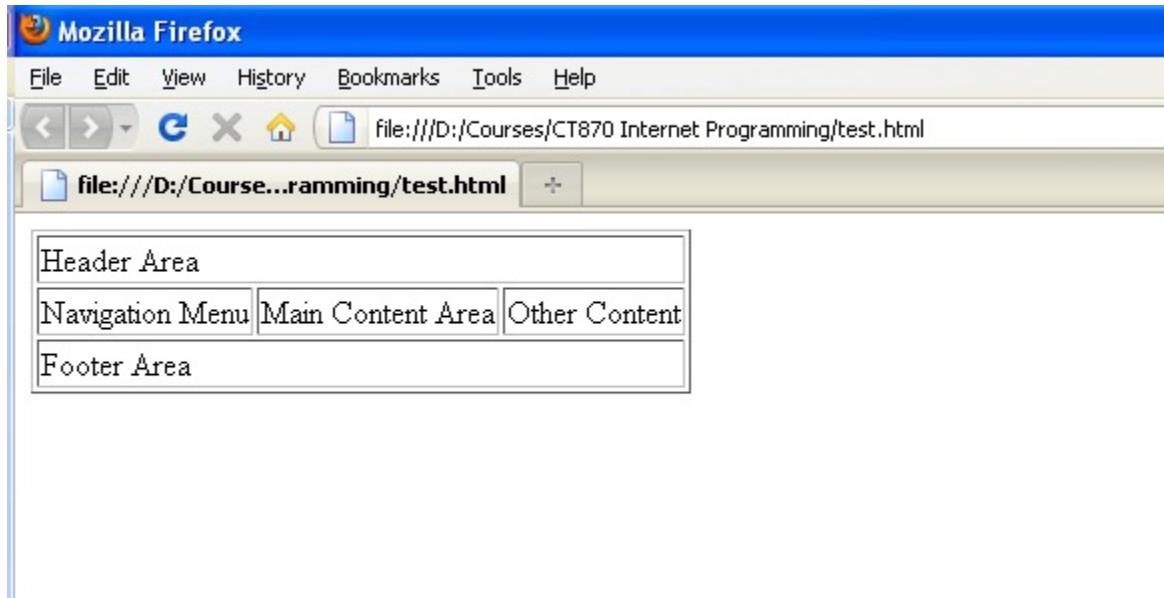


This can be achieved using an HTML Table

# Coding the Table

```
<table border="1">
<tr>
    <td colspan="3">Header Area</td>
</tr>
<tr>
    <td>Navigation Menu</td>
    <td>Main Content Area</td>
    <td>Other Content</td>
</tr>
<tr>
    <td colspan="3">Footer Area</td>
</tr>
</table>
```

# Table within Firefox



- Table occupies only a corner of the screen
- Can use `width` attribute to force dimensions

# Sizing the table

Can set width in two ways:

- By Pixels

- By Percentage of Available Width

Must consider screen resolutions when picking a width

Device resolutions range from less than 800x600 pixels (mobile phone) to in excess of 1920x1080 pixels (high resolution desktop PC)

# Setting the Width

Consider the following table dimensions

Width = 950		
Width = 200	Width = 550	Width = 200
Width = 950		

# Coding the Table – Adding Width

```
<table border="1">
<tr>
    <td colspan="3" width="950">Header Area</td>
</tr>
<tr>
    <td width="200">Navigation Menu</td>
    <td width="550">Main Content Area</td>
    <td width="200">Other Content</td>
</tr>
<tr>
    <td colspan="3" width="950">Footer Area</td>
</tr>
</table>
```

# Wide Table – Needs Height



- Table is now wide across the screen
- Can use height attribute to fill it out

Header Area	
<b>Navigation Menu</b> <ul style="list-style-type: none"><li>• <a href="#">Link 1</a></li><li>• <a href="#">Link 2</a></li><li>• <a href="#">Link 3</a></li><li>• <a href="#">Link 4</a></li></ul>	<b>Main Content Area</b> <p>This is some sample text to demonstrate the sample page using the table created with HTML. This text should fill up the main content area a bit and help illustrate the shape that we get from using the table.</p>
	<b>Other Content</b> <p>This is some sample text to demonstrate the sample page using the table created with HTML. This text should fill up the main content area a bit and help illustrate the shape that we get from using the table.</p>
<b>Footer Area</b>	

# Exercise (not graded)

Make a webpage which displays Looney Tunes characters in a table. Recall from a few weeks ago, there are 6 images available at these URLs:

<http://www.psychicsoftware.com/ct1113/banner-image-1.png>

<http://www.psychicsoftware.com/ct1113/banner-image-2.png> etc.

And, the top banner image is here:

<http://www.psychicsoftware.com/ct1113/topbanner-looneytunes.jpg>



# **CT1114**

# **Web Development**

## **Section 9:**

## **Introducing CSS**

---

Dr. Sam Redfern

[sam.redfern@nuigalway.ie](mailto:sam.redfern@nuigalway.ie)

@psychicsoftware



NUI Galway  
OÉ Gaillimh

# Discord / Qwickly



Use our Discord channel for questions, comments etc. at any time

<https://discord.gg/W9rE546>

**Qwickly Code (Attendance Monitoring)**

# Reminder: HTML

HyperText Markup Language

The language used to define the contents of web pages  
Includes text, tables, hyperlinks, images, etc.

HTML is the language used to define webpage CONTENT  
and STRUCTURE

It is normally downloaded by your browser from a web server, and then interpreted and displayed (rendered) by your browser

# CSS

## Cascading Style Sheets

The language used to define the style of web pages  
Includes colours, fonts, sizes, positions, etc.

CSS is the language used to define webpage **VISUAL  
STYLE**

It is normally downloaded by your browser from a web server, and then interpreted by your browser and used to define the visual style of an HTML page that references it

# Cascading Style Sheets (CSS)

- CSS allows the developer to define a set of styles
- These styles can be used across multiple pages
- Helps to create a uniform and consistent look across the website
- Makes formatting content much easier

# Syntax of CSS

Styles are composed of two parts:

**Selector**

**Declarations**

The **Selector** is the HTML element involved  
`<p>`, `<h1>`, etc

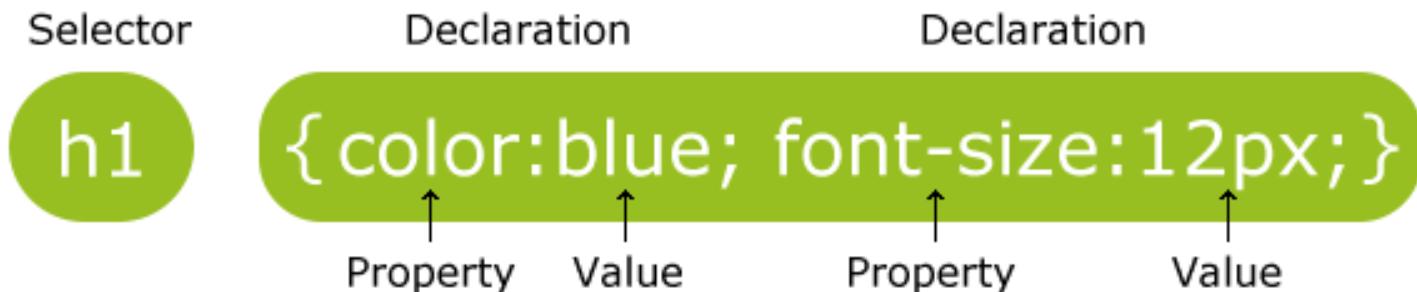
The **Declaration** consists of two parts  
a **Property** to be affected (color, size, etc)  
a **Value** to set it to (red, 12, etc)

# Syntax – Continued

Example of styling H1 tags

Colour them Blue

Set their size to 12 pixels



# Syntax – Continued

```
h1  
{  
color: blue;  
font-size: 12px;  
}
```

## Note:

color **not** colour

Braces { } separate the  
selector from the  
declarations

Each declaration ends in  
semi-colon ;

# Syntax – Continued

```
h1
{
color: blue;
font-size: 12px;
}
```

## Note:

**No space** between value  
and units (12 and px)  
**No quotes** for values

# CSS Styling

Some parameters that can be styled

# Background Colour

Styled by using

`background-color:value`

Accepts values of HTML Hex Colour Codes

Also standard colour names – red, blue, etc

Applicable to many elements:

Headings

Paragraphs

Document

Etc.

# Text Formatting

Colour of text can be set using  
color:value (Note US spelling)

Alignment can be set using

text-align:left

text-align:right

text-align:**center**

**(Note US spelling)**

text-align:justify

# Text Fonts

Font face specified in groups of fonts using

font-family:value

If the user doesn't have the first font, the second font is used (from comma-separated list), etc.

Final font should reference a generic family of fonts

serif, sans-serif or monospace

```
h1 {font-family:Arial, Verdana, sans-serif; }
```

Multiple fonts separated by commas

Times New Roman is a serif font

# Serif vs. Sans Serif

Serif fonts are those like Times New Roman

## Serif Font

Serif are usually used in print

Sans Serif fonts are those like Arial

## Sans Serif Font

Sans Serif usually display better on PCs

# Styling Links

Links (anchor tags) can be styled like any other text  
colour, size, background-color, font, etc

Links also have four special selectors

```
a:link {color:value;}  
a:visited {color:value;}  
a:hover {color:value;}  
a:active {color:value;}
```

*Unvisited Link*  
*Visited Link*  
*Mouse over Link*  
*Selected Link*

**Must follow the above order**

# Example

Here is some text inside a paragraph, and **some of it** is also inside a span.

```
<!DOCTYPE html>
<html>
  <head>
    <title>CSS Simple Example</title>
    <style>
      p {
        color: blue;
        font-size: 14px;
      }
      span {
        color: #FF00FF;
        font-size: 18px;
      }
    </style>
  </head>
  <body>
    <p>Here is some text inside a paragraph, and <span>some of it</span> is also
    inside a span.</p>
  </body>
</html>
```

# Exercise

## Kubla Khan

In Xanadu did Kubla Khan  
A stately pleasure-dome decree:  
Where Alph, the sacred river, ran  
Through caverns measureless to man  
Down to a sunless sea.

```
<html>
  <head>
    <title>CSS Exercise</title>
  </head>
  <body>
    <h2>Kubla Khan</h2>
    <p>
      In Xanadu did Kubla Khan<br>
      A stately pleasure-dome decree:<br>
      Where Alph, the sacred river, ran<br>
      Through caverns measureless to man<br>
      Down to a sunless sea.
    </p>
  </body>
</html>
```

Take this HTML code and add styles as indicated in the picture above

# Placement of your Style Code

So far, we have been doing Embedded Style, where our style code is embedded in the `<head>` section of a document

CSS actually provides 3 different places to define style, which gives a lot of flexibility..

# Three Places to Define Styles

## Inline Style

Style is defined for a single element using its **style attribute**

## Embedded Style

Block of CSS inside the document Head: applies to that specific page only

## External Style Sheets

Separate (external) file containing styles to be used across all pages that reference the sheet

# Effect of Style Placement

## Inline Style

Affects only the element that it's applied to  
Localised, highest priority

## Embedded Style

Affects only the page that it's embedded on  
Medium priority

## External Style Sheet

Globally affects all pages calling it  
Lowest priority

# Prioritisation of Style Types

The higher priority style type will supersede the lower priority style type

## Example: Styling a Heading

Inline style tells the heading to be size 20

External style says it should be size 18

Inline style has higher priority (as it's localised) and sets the size to be 20

This process is called *Cascading*  
Hence, **Cascading Style Sheets**

# Inline Styles

```
<p style="color:blue; font-size:10px;">Text</p>
```

Uses `style` attribute for a single HTML element

Style is added *in line* with the HTML

Affects only that element

Others of the same type unchanged

To be used sparingly

External sheets offer more global consistency

Too many inline styles defeat the purpose, and is "bad software engineering".

Generally we'd prefer to have a *separation of concerns*, and inline styles are poor in this regard

# Embedded Styles

Embedded in the Head of the HTML page

Affects only the page containing the style

Takes priority over external style sheets

Less priority than inline styles

```
<head>
<style>
  h1 { font-size:30px; }
  h2 { font-size:20px; }
  body
  {background-color:#CCCCCC; }
</style>
</head>
```

# External Style Sheets

Separate external file with **.css** extension

Link to the sheet is added to the Head of each page that should use the stylesheet

```
<head>
<link rel="stylesheet" type="text/css"
      href="mystyle.css" />
</head>
```

# External Style Sheets

Must not contain any HTML

Simply lists the selectors and declarations required

Style Sheets can be commented

`/* Comment goes here */`

Just like in C, C++, Java, etc

**NB:** Double forward slash ( // ) comments don't work

# Exercise

- Three paragraphs have been coloured using inline styling
- Your job is to write Javascript so that when a paragraph is clicked, its style.color is set to "red", while the other two paragraphs have their style.color set to "green"
- See initial code on next slide (and via Discord)

## Kubla Khan

In Xanadu did Kubla Khan  
A stately pleasure-dome decree:  
Where Alph, the sacred river, ran  
Through caverns measureless to man  
Down to a sunless sea.

So twice five miles of fertile ground  
With walls and towers were girdled round;  
And there were gardens bright with sinuous rills,  
Where blossomed many an incense-bearing tree;  
And here were forests ancient as the hills,  
Enfolding sunny spots of greenery.

But oh! that deep romantic chasm which slanted  
Down the green hill athwart a cedar cover!  
A savage place! as holy and enchanted  
As e'er beneath a waning moon was haunted  
By woman wailing for her demon-lover!  
And from this chasm, with ceaseless turmoil seething,  
As if this earth in fast thick pants were breathing.

```
<!DOCTYPE html>
<html>
<head>
<title>CSS Exercise</title>
</head>
<body>
<h2>Kubla Khan</h2>
<p style="color:red;">
In Xanadu did Kubla Khan<br>
A stately pleasure-dome decree:<br>
Where Alph, the sacred river, ran<br>
Through caverns measureless to man<br>
Down to a sunless sea.
</p>
<p style="color:green;">
So twice five miles of fertile ground<br>
With walls and towers were girdled round;<br>
And there were gardens bright with sinuous rills,<br>
Where blossomed many an incense-bearing tree;<br>
And here were forests ancient as the hills,<br>
Enfolding sunny spots of greenery.
</p>
<p style="color:green;">
But oh! that deep romantic chasm which slanted<br>
Down the green hill athwart a cedarn cover!<br>
A savage place! as holy and enchanted<br>
As e'er beneath a waning moon was haunted<br>
By woman wailing for her demon-lover!<br>
And from this chasm, with ceaseless turmoil seething,<br>
As if this earth in fast thick pants were breathing.
</p>
</body>
</html>
```

# Graded Exercise

You are tasked with creating a website for a fictitious online shop. You are to name this business and also define what it is that they sell.

Each page should be laid out using an invisible (border=0) table, and contain a **header and footer**, a **navigation area** and a **content area**. The navigation area should contain links to each of the three pages in the website (using hyperlink anchors <a>).

The website should have three separate pages:

- **Home (home.html)**
  - This should be the main landing page of your website and should show the latest news and updates from your business.
- **About Us (about.html)**
  - This page should explain to a new user what your business sells as well as its mission statement.
- **Contact Information (contact.html)**
  - Use a **list** to give the contact information for the business' Fulfilment Executive, Marketing Officer and Secretary. Each person should have their name and phone number listed.

## Styling

- You should use an external stylesheet (i.e., a .css file separate from the .html files) to define styles for all three pages of the website. Include in this at least 4 style definitions, of your choice.
- Submit your code as a .zip archive, via Blackboard. It should consist of three .html files and one .css file.

# CT1114

# Web Development

## Section 10:

## Timed Code and Animation in Javascript

Dr. Sam Redfern

[sam.redfern@nuigalway.ie](mailto:sam.redfern@nuigalway.ie)

@psychicsoftware



NUI Galway  
OÉ Gaillimh

# window.setTimeout()

Schedule the deferred execution of a piece of Javascript code (often, the code takes the form of a call to a function)

Delay is specified in milliseconds (i.e. thousandths of a second)

Useful e.g. for controlling animation or other special effects which require the gradual modification of an element's attributes (e.g. position, colour) as time passes.

You can specify an arbitrary piece of Javascript (in quotes) to execute:

```
window.setTimeout("alert('hello')", 1000);
```

Or you can specify a function (by name) to execute. Note that the function name only is used (no brackets after it). That's because you're actually sending the function as an argument, not writing a line of code that's executing it:

```
window.setTimeout/animate, 50);
```

(In Javascript, functions are referred to as "first class objects".. We'll see how useful it is to send functions as arguments, later..)

# Example: countdown timer using a textbox

```
<html>
<head>
<script>
function countDown() {
    var oBox = document.getElementById("currValue");
    var v = parseInt(oBox.value);
    if (v>=1) {
        v--;
        oBox.value = v;
        window.setTimeout(countDown, 1000);
    }
    if (v<=0) {
        oBox.value = "Lift off!";
    }
}
</script>
</head>
<body>
Countdown!<br>

<input id='currValue' value='10'> <input type=button value='Go'
    onClick='countDown();'>

</body>
</html>
```

# Problem?

What happens if we click the 'Go' button more than once, in the previous example?

What can we do to fix that?

# The Date Object

- A useful object that provides the current date and time at the instant at which it is created
- To create a Date object:

```
var myDate = new Date();
```

- Methods of the Date object:

.getFullYear()	- gives year as 4 digit integer
.getMonth()	- gives month number as 0-11
.getDate()	- gives day number of month as 1-31
.getDay()	- gives day number in week as 0-6, (0=Sunday)
.getHours()	- gives hour of the day as 0-23
.getMinutes()	- gives elapsed minutes in current hour, 0-59
.getSeconds()	- gives elapsed seconds in current min, 0-59

# Exercise

18:06:53

- Make an onscreen clock that updates once per second, displaying hour, minutes, and seconds
- You can start with this:

```
<!DOCTYPE html>
<html>
<head>
<script>
    function updateClock() {
        // needs code here!
    }
</script>
<body onload="updateClock();">
    <p id="theClock" style="text-align:center; font-size:120px;">00:00:00</p>
</body>
```

# <div> tags and Absolute Position

<div> tags are very useful for creating sections of content that can be positioned precisely on the page irrespective of anything else on the page (to do this, set the 'position' property of their style to 'absolute' and define the 'left' and 'top' properties of their style to some pixel values e.g. "300px")

Analogy: sheets of tracing paper laid on top of a page can have content on them and can be slid around on top of the page

E.g. a div with an image in its innerHTML can be moved by changing its style.left and style.top :

```
<div id='divVan' style='position:absolute; left:800px; top:100px'>
  <img src='mysteryMachine.png'>
</div>
```

# Animation with window.setTimeout

```
<html>
<head>
<script>
    var leftPos=800;
    function moveTheVan() {
        leftPos -= 2;
        document.getElementById("divVan").style.left = leftPos+"px";
        if (leftPos>0)
            window.setTimeout(moveTheVan, 20);
    }
</script>
</head>
<body>
    <div id='divVan' style='position:absolute; left:800px; top:100px'>
        <img src='mysteryMachine.png'>
    </div>
    <button onClick='moveTheVan();'>Scoobie Doo!</button>
</body>
</html>
```



Question: why haven't we just done the movement using a 'while' or 'for' loop in place of window.setTimeout() ?

# Two separate animations

```
<html>
<head>
<script>
var leftPos=400;
var speed=-4;
var bugImg = "bug2_1.jpg";

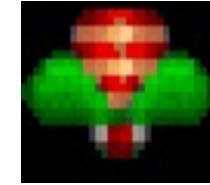
function move() {
    leftPos += speed;
    if (leftPos<=5 && speed<0)
        speed=4;
    else if (leftPos>=595 && speed>0)
        speed=-4;

    document.getElementById("divBug").style.left = leftPos + "px";
    window.setTimeout(move, 20);
}

function flapWings() {
    if (bugImg=="bug2_1.jpg")
        bugImg="bug2_2.jpg";
    else
        bugImg="bug2_1.jpg";

    document.getElementById("imgBug").src = bugImg;
    window.setTimeout(flapWings, 1000);
}

</script>
</head>
<body style="background-color:black;" onload="move(); flapWings();">
<div id='divBug' style='position:absolute; left:400px; top:100px'>
    <img id='imgBug' src='bug2_1.jpg'>
</div>
</body>
</html>
```



# Some math(s) functions

- `Math.random()`
  - Returns a random number between 0.0 and 0.999999
- `Math.floor(x)`
  - Returns x rounded down to the nearest integer
- `Math.round(x)`
  - Returns x rounded up or down to the nearest integer
- `Math.cos(x)`   `Math.sin(x)`
  - Returns the Cosine/Sine of x (angle in radians)
- And lots more..
  - [https://www.w3schools.com/js/js\\_math.asp](https://www.w3schools.com/js/js_math.asp)

# Graded Exercise

- Display an <img> inside a <div>
- Position the <div> at a random place on the document
- Hint:

```
var x = Math.random() * 600;
```

- x now has a random number between 0 and 600, which can be used as the value for the .style.left property of the <div>
- Every time the <div> is clicked, move it to a different random place on the document

# CT1114

# Web Development

## Section 11:

## More Javascript Animation and Events

Dr. Sam Redfern

[sam.redfern@nuigalway.ie](mailto:sam.redfern@nuigalway.ie)

@psychicsoftware



NUI Galway  
OÉ Gaillimh

# Last Week's Graded Exercise

- Display an `<img>` inside a `<div>`
- Position the `<div>` at a random place on the document
- Hint:

```
var x = Math.random() * 600;
```

- `x` now has a random number between 0 and 600, which can be used as the value for the `.style.left` property of the `<div>`
- Every time the `<div>` is clicked, move it to a different random place on the document

# Exercise

- Starting with the ‘animated mystery machine van’ code from section 10, modify it so that the van does not start moving until it is clicked.
- Further modify the code so that the van stops moving if it is clicked a second time.
- Questions:
  - What extra behaviour are we adding here?
  - What’s an appropriate way to store the data necessary to achieve this?
- (See next slide for initial code)

# Animation with window.setTimeout

```
<html>
<head>
<script>
    var leftPos=800;
    function moveTheVan() {
        leftPos -= 2;
        document.getElementById("divVan").style.left = leftPos+"px";
        if (leftPos>0)
            window.setTimeout(moveTheVan, 20);
    }
</script>
</head>
<body>
    <div id='divVan' style='position:absolute; left:800px; top:100px'>
        <img src='mysteryMachine.png'>
    </div>
    <button onClick='moveTheVan(); '>Scoobie Doo!</button>
</body>
</html>
```



Question: why haven't we just done the movement using a 'while' or 'for' loop in place of window.setTimeout() ?

# Javascript event handlers

Recall that we have used some events and event handlers already, e.g:

```
<body onload="giveChoice();">
```

```
<button onclick="showFactors();>Show Factors</button>
```

```
<textarea id='txtUserInput' onkeyup="updateOutput();>
```

We'll now introduce some more... (slides below)

# Loading and Unloading a Page

- **Events on the <body> tag:**
- **onload**
  - Triggered when a page has *\*finished\** loading
  - *When the page loads in the browser*
- **onunload**
  - Triggered when a user exits a page
  - *When the page is unloaded from the browser*
- **onload** is also used by the <img> tag... question: why might that be?

# Selecting and De-Selecting Elements

- All three normally used with form elements such as `<input>`, `<textarea>`
- **onfocus**
  - Triggered when an element gets the keyboard focus
  - an element that is clicked on is said to be "in focus"
- **onblur**
  - Triggered when an element loses focus
- **onchange**
  - Triggered when the content of an element changes

# As the Mouse Moves Over HTML Elements

- **onmouseover**
  - Triggered for an element when the mouse cursor is moved over that element
  - e.g. moving the mouse over an image ('rollover') can be used to change the image's src property (see example below)
- **onmouseout**
  - Triggered for an element when the mouse cursor is moved away from that element
  - e.g. moving the mouse out of the image can be used to revert the image's src property to its initial value
- **onmousemove**
  - Triggered every time the mouse moves while over an element
  - Receives the mouse's current position on the element as an argument (see example below)

# Other Mouse Events

- **onclick**
  - Triggered when the mouse clicks an element
- **ondblclick**
  - Triggered when the mouse double clicks an element
- **onmousedown**
  - Triggered when the mouse button is pressed on an element
- **onmouseup**
  - Triggered when the mouse button is released, having previously been pressed on an element

# onmousemove Example

```
<!DOCTYPE html>
<html>
<head>
<script>
    function showCoords(e) {
        document.getElementById("pOutput").innerHTML =
            e.clientX+", "+e.clientY;
    }
</script>
</head>
<body>
    <div style="width:600px; height:600px; background-
color:#BBBBBB;" onmousemove="showCoords(event);">
        <p id="pOutput"></p>
    </div>
</body>
</html>
```

The mousemove event receives an object (e) which has various useful information in it such as e.clientX and e.clientY

# Exercise

- Display an `<img>` in a document
- Display in a `<p>` how many times the `<img>` has been clicked and doubleclicked, and how many times mouseover and mouseout have happened to the `<img>`.



Clicks: 22

Doubleclicks: 8

Mouseovers: 16

Mouseouts: 16

# Example: image follows mouse

```
<!DOCTYPE html>
<html>
<head>
<script>
    function moveTheVan(e) {
        var oImg = document.getElementById("imgVan");
        oImg.style.left = e.clientX-(oImg.naturalWidth/2)+"px";
        oImg.style.top = e.clientY-(oImg.naturalHeight/2)+"px";
    }
</script>
</head>
<body onmousemove="moveTheVan(event);">
<div style="width:100%; height:100%; background-color:#888888;
position:absolute; left:0px; top:0px;">
    <img id='imgVan' style='position:absolute; left:0px;
top:0px;' src='mysteryMachine.png'>
</div>
</body>
</html>
```

# Exercise

- Modify the previous example so that the van only follows the mouse if the mouse button is held down on it
  - i.e. implement a ‘drag’ behaviour for the image
- Questions:
  - What additional events do we need to handle, for this?
  - Which element should these events be attached to? (`onmousemove` was attached to the `<body>`)
  - What additional data do we need?

# Other Keyboard Events

- **onkeydown**
  - Triggered when a keyboard key is pressed
- **onkeyup**
  - Triggered when a keyboard key is released
- **onkeypress**
  - Triggered when a keyboard key is pressed or held
- *For each of these, an object (e) is received as an argument, and e.keyCode identifies the key which has triggered the event*

# A (Stylish) Events Handling Example

Sets the background colour of text fields to red if their content is invalid, and white if it's ok

```
<!DOCTYPE html>
<html>
<head>
<script>
function validateYear() {
    var oYear = document.getElementById('year');
    var y = parseInt(oYear.value);
    if (isNaN(y) || y<1900 || y>2019)
        oYear.style.background = "#FF0000";
    else
        oYear.style.background = "#FFFFFF";
}
function validateEmail() {
    var oEmail = document.getElementById('eMail');
    var e = oEmail.value;
    if (e.indexOf("@")<0 || e.indexOf(".")<0) {
        oEmail.style.background = "#FF0000";
    }
    else {
        oEmail.style.background = "#FFFFFF";
    }
}
</script>
</head>
<body>
Please enter your year of birth:
<input type="text" id="year" value="1980" size="10" onkeyup="validateYear();"><br><br>
Please enter your email address:
<input type="text" id="eMail" value="" size="10" onkeyup="validateEmail();"><br><br>
</body>
</html>
```

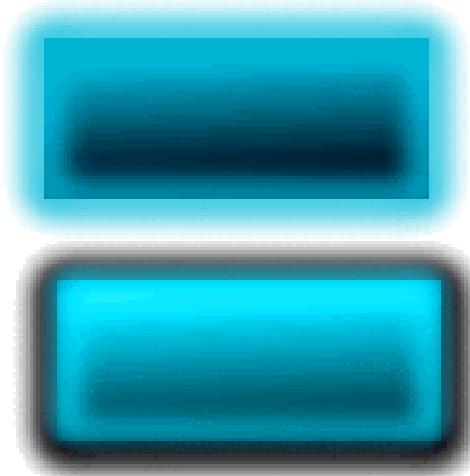
# Events example: 'rollover button' image

```
<html>
<head>
<script>
function preloadImages() {
    var Image1 = new Image();
    Image1.src = "btnOver.gif";
}
function changeImgSrc(oImg, sSrc) {
    oImg.src = sSrc;
}
</script>
</head>
<body onLoad="preloadImages();">

</body>
</html>
```

Preloading is a nice touch  
but not strictly necessary

Notice the 'this' argument which  
provides a reference to the  
object receiving the event -- therefore  
we can pass this to changeImgSrc()  
and therefore don't need to use  
document.getElementById to get it in  
changeImgSrc()



# Exercise (not graded)

```
<html>
<head>
<script>
var leftPos = 800, topPos = 100;

function init() {
    animate();

    window.onkeydown = function(e) {
        alert("You pressed: "+e.keyCode);
    };

    window.onkeyup = function(e) {
    };

}

function animate() {
    document.getElementById("divVan").style.left = leftPos+"px";
    document.getElementById("divVan").style.top = topPos+"px";
    window.setTimeout(animate, 20);
}
</script>
</head>
<body onload="init();">
<div id='divVan' style='position:absolute; left:800px; top:100px'>
    <img src='mysteryMachine.png'>
</div>
</body>
</html>
```

- Make an image move left, right, up, and down when the arrow keys are pressed
- Note the slightly different way of attaching events to objects
- Start with this code

# CT1114

# Web Development

## Section 12:

## Arrays

Dr. Sam Redfern

[sam.redfern@nuigalway.ie](mailto:sam.redfern@nuigalway.ie)

@psychicsoftware



NUI Galway  
OÉ Gaillimh

# Arrays in Javascript

- Arrays in Javascript are similar to those in other languages such as C.
- An array stores multiple values using a single array name, with a value supplied in square-brackets defining the index which you wish to access
  - The index value you supply in square-brackets often comes from the contents of a variable
  - This means you can write generic code to deal with any element in an array, rather than hard-coding it to a specific variable
  - Dealing with 1 million elements is as easy as dealing with 1 element
- An array is a 'random access' data structure
  - i.e. you can read/write any elements at any time without needing to do so in any specific sequence
- In several ways, Javascript provides more powerful arrays than C:
  - arrays can be 'sparse' i.e. have gaps (which is useful)
  - arrays can be indexed on strings as well as integers (which is useful)
  - each array element can contain any type of data (numbers, strings, even more arrays, objects, or *functions!* – also useful!)

# Arrays in Javascript

```
var a = [ ];           // create an empty array
a.push(22);          // add an element to the end (i.e. at index 0 in this case)
a[20] = "test";       // set an element at index 20 (now, indices 1-19 are undefined)
a.splice(10,1);       // removes one element from index 10 (and shifts others down to fill
                     // the gap)
a[15] = [6,7,8];     // set an element at index 15 (this element is itself an array, so for
                     // example the value of a[15][0] is now 6)
a["exit"] = "salida"; // set an element at index "exit".. is this useful?
a.length             // returns the index of the last element, assuming the array is being
                     // indexed by integers rather than strings
```

```
if (a[15] != undefined)
  alert("the array has something defined at index 15");
else
  alert("the array is empty at index 15");
```

Note that any variables in Javascript which do not exist have the special value ***undefined***

```
if (a["exit"] != undefined)
  alert("the Spanish for 'exit' is "+a["exit"]);
else
  alert("I don't know the Spanish for 'exit'");
```

# Use of Undefined

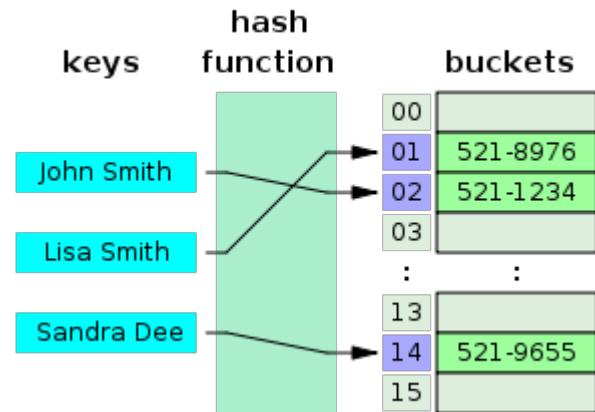
```
<html>
<head>
<script>
    var a = [];
    a["exit"] = "salida";

    function translate(englishWord) {
        if (a[englishWord] != undefined)
            alert("the Spanish for '" +englishWord+ "' is
"+a[englishWord]);
        else
            alert("I don't know the Spanish for
'" +englishWord+ "'");

    }
</script>
<head>
<body onload="translate('exit'); translate('hello');">
</body>
</html>
```

# Hash Tables

A hash table is a data structure used to implement an associative array, a structure that can map keys to values. A hash table uses a hash function to compute an index into an array of buckets or slots, from which the desired value can be found.



.. In Javascript we can do this:

```
var tel = [];
tel["John Smith"] = "521-1234";
tel["Lisa Smith"] = "521-8976";
tel["Sandra Dee"] = "521-9655";
```

In fact, in Javascript (unlike in C) arrays are implemented internally as Hash Tables.

# Iterating arrays

- Assuming that we have an array called 'a':

- Normal (dense, integer-indexed) arrays:

```
for (var i=0; i<a.length; i++)  
    txt += "Element "+i+" is "+a[i]+"<br>;
```

- Sparse (or string-indexed) arrays:

```
for (var i in a)  
    txt += "Element "+i+" is "+a[i]+"<br>;
```

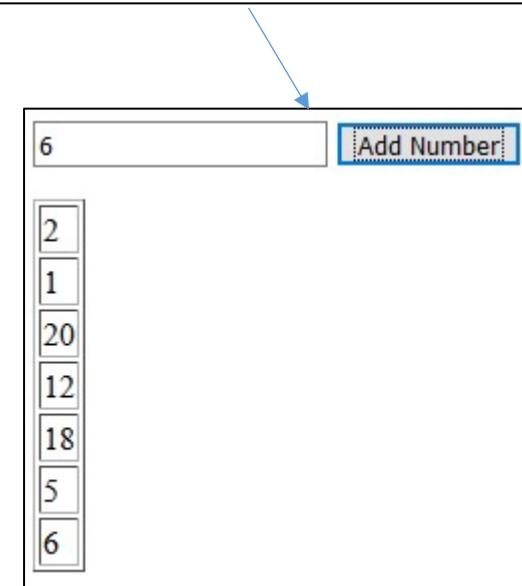
# Exercise

```
<!DOCTYPE html>
<html>
<head>
<script>
    var nums = [ ];

    function addNumber() {
        var txt = document.getElementById("txtNumber").value;
        var number = parseInt(txt);
        if (isNaN(number))
            alert(txt+" is not a number!");
        else {
            nums.push(number);
            redrawTable();
        }
    }

    function redrawTable() {
        // write your code here!
    }
</script>
</head>
<body>
    <input type='text' id='txtNumber'> <button onclick='addNumber();'>Add Number</button>
    <p id="pOutput"></p>
</body>
</html>
```

- The code below adds a new number to the end of the 'nums' array every time the user clicks the 'Add Number' button
- Modify this code so that it also displays, in a table in the paragraph 'pOutput', the numbers entered (as shown here)



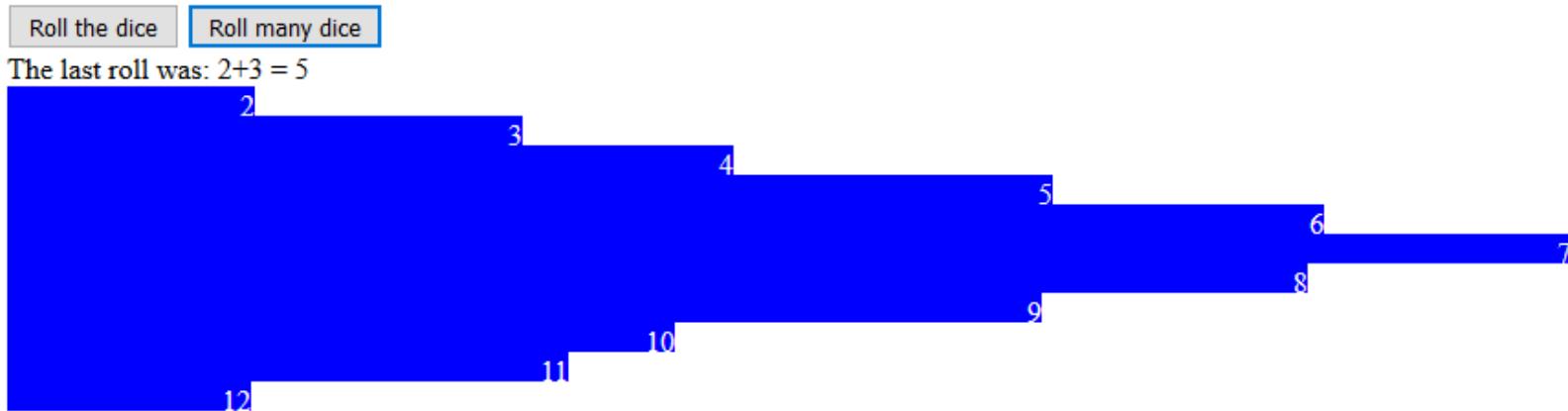
# Exercise

- Modify the previous program so that each number has a ‘delete’ button beside it
- When a ‘delete’ button is clicked, the corresponding number should be remove from the array (using `nums.splice`), and the table of data should be redrawn
- Question: how do we make delete buttons that are associated with particular array elements?

The image shows a user interface for managing a list of numbers. At the top left is a text input field containing the number '7'. To its right is a button labeled 'Add Number'. Below these are five rows of data, each consisting of a number and a 'Delete' button. The numbers are 999, 34, 64, 2, and 7, arranged vertically. Each number is in a white box, and each 'Delete' button is in a blue box.

999	Delete
34	Delete
64	Delete
2	Delete
7	Delete

# Example: dice rolling, arrays, and making a bar chart with <div> tags



```
<html>
<head>
    <script src='12c-barcharts.js'></script>
</head>
<body>
    <button onclick='roll();'>Roll the dice</button>
    <button onclick='rollMany();'>Roll many dice</button><br>
    The last roll was: <span id='lastRoll'></span><br>
    <div id='barChart'></div>
</body>
</html>
```

```
// this file is 12c-barcharts.js
var frequency = [];
for (var i=2;i<=12;i++)
    frequency[i] = 0;

function roll() {
    var dice1 = 1 + Math.floor(Math.random()*6);
    var dice2 = 1 + Math.floor(Math.random()*6);
    var sum = dice1+dice2;
    document.getElementById("lastRoll").innerHTML = dice1 + " +" + dice2 + " = " + sum;

    frequency[sum]++;
    updateBarChart();
}

function rollMany() {
    var rollTimes = parseInt(window.prompt("How many rolls?"));
    var dice1, dice2;
    if (rollTimes>0) {
        for (var i=0;i<rollTimes;i++) {
            dice1 = 1 + Math.floor(Math.random()*6);
            dice2 = 1 + Math.floor(Math.random()*6);
            frequency[dice1+dice2]++;
        }
        document.getElementById("lastRoll").innerHTML = dice1 + " +" + dice2 + " = " +
(dice1+dice2);
        updateBarChart();
    }
}

function updateBarChart() {
    var html = "";
    for (var i=2;i<=12;i++) {
        html += "<div style='background:blue; width:"+frequency[i]+"px; height:16px; margin:0px;
color:white; text-align:right;'>" + i + "</div>";
    }
    document.getElementById("barChart").innerHTML = html;
}
```

# Graded Exercise

- Make a web page which, when it loads, performs the following:
- Creates an array containing 100 random integers, each between 1 and 999.
- Displays, in a `<p>`, the sum, average, largest and smallest value from the set of numbers.
- Displays, in another `<p>`, the numbers themselves, comma separated

# CT1114

# Web Development

## Section 13:

## Splitting Strings, Sorting Arrays

Dr. Sam Redfern

[sam.redfern@nuigalway.ie](mailto:sam.redfern@nuigalway.ie)

@psychicsoftware



NUI Galway  
OÉ Gaillimh

# Last Week's Graded Exercise

- Create an array containing 100 random integers, each between 1 and 999.
- In a `<p>`, display the sum, average, largest and smallest value from the set of numbers.
- Display the numbers themselves, comma separated, in another `<p>`

# String method: .split(" ")

- Splits a string into an array of substrings.
- Splits are made according to the delimiter argument (which is often a space or \t character)

```
<html>
<head>
<script>
function splitWords() {
    var txt = document.getElementById("userText").value;
    var arr = txt.split(" ");
    var output = arr.length + " words";
    for (var i=0;i<arr.length;i++)
        output += "\n" + arr[i];
    document.getElementById("outputText").value = output;
}
</script>
</head>
<body>
    Type some stuff..<br>
    <input id='userText' onkeyup='splitWords();'><br>
    <textarea id='outputText'></textarea>
</body>
</html>
```

# Array method: `.sort()`

- Sorts an array
  - With no argument => sort alphabetically (even if you have an array of numbers)
  - With an argument => sort using a custom function defining how to order two elements (see next slide)
- E.g. modify previous example:

```
function splitWords() {  
    var txt = document.getElementById("userText").value;  
    var arr = txt.split(" ");  
    var output = arr.length + " words";  
    arr.sort();  
    for (var i=0;i<arr.length;i++)  
        output += "\n" + arr[i];  
    document.getElementById("outputText").value = output;  
}
```

- But what if we typed "1 2 12 15 6" ? And why do we see this behaviour?

# Array method: .sort(compareFunction)

- You send a function as the argument (name only, no round brackets since you're not executing it).
- **compareFunction** is called whenever the (behind-the-scenes) sorting algorithm needs to test two elements
- **compareFunction** is sent the two elements as arguments
- **compareFunction** should return a negative number if the 1<sup>st</sup> argument is lower, a positive number if the 2<sup>nd</sup> argument is lower, and zero if they're the same
- Technically, in this context, compareFunction is referred to as a 'callback' - we're sending it as an argument to another function.  
*We're adding our own custom code into someone else's library function, without needing to even see their function's code.*
- Yes, that's pretty cool.
- E.g... see next slide

```
<script>

function splitWords() {
    var txt = document.getElementById("userText").value;
    var arr = txt.split(" ");
    var output = arr.length + " words";
arr.sort(compare);
    for (var i=0;i<arr.length;i++)
        output += "\n" + arr[i];
    document.getElementById("outputText").value = output;
}

function compare(x,y) {
    x = parseInt(x);
    y = parseInt(y);
    if (x<y)
        return -1;
    else if(y<x)
        return 1;
    else
        return 0;
}

</script>
```

# Exercise

984,209,501,428,7,313,990,439,855,38,375,769,488,491,188,677,110,300,711,425,246,637,483,630,170,460,954,172,359,614,920,21,13,232,707,364,424,724,929,341,530,571,860,864,784,156,729,965,979,370,818,275,425,185,23,51,346,136,305,869,207,572,669,762,343,644,302,412,873,273,686,630,693,285,250,416,739,757,160,484,363,106,240,547,546,521,954,666,707,646,597,256,730,152,755,914,728,746,384,522,149,905,932,339,711,681,887,486,586,994,591,321,103,527,959,214,311,452,291,2,272,200,653,312,263,220,607,245,987,396,879,659,238,634,972,869,87,755,770,923,230,9,104,4,285,238,673,243,284,523,778,108,634,44,687,26,605,888,302,283,470,832,577,475,634,332,897,715,393,137,309,990,305,482,572,946,696,158,97,785,62,653,704,345,968,628,406,936,804,996,258,375,35,617,202,698,144,346,495,67,23,213,261,212,154,279,847,726,430,641,113,413,33,840,711,4,123,919,943,191,167,381,393,608,828,4,393,708,836,75,59,907,792,497,522,331,991,861,668,674,741,773,450,903,684,239,166,535,824,687,636,247,916,356,300,606,494,493,559,543,179,992,201,831,996,377,530,164,331,876,914,274,398,948,119,408,413,785,530,972,919,282,724,276,243,345,217,727,783,382,701,365,207,181,45,215,593,815,297,664,466,959,528,904,998,165,33,784,414,59,873,808,202,168,819,485,783,131,854,865,763,912,245,249,614,435,697,152,649,893,21,677,152,573,366,502,443,119,386,350,158,292,197,911,456,697,760,951,414,775,869,744,83,169,506,129,402,277,118,537,701,865,450,85,704,140,723,275,862,135,422,120,432,381,426,589,282,422,116,945,754,552,783,451,212,420,171,705,217,949,98,695,912,204,217,172,697,228,665,121,412,51,550,13,196,231,19,537,251,285,154,630,538,905,173,178,641,789,21,998,620,344,284,102,919,455,350,631,700,951,180,348,184,207,733,525,844,578,251,356,721,432,716,81,990,101,473,957,296,964,158,348,63,954,424,629,264,210,414,209,299,670,275,676,601,985,975,626,819,493,988,151,30,5,113,906,891,956,925,920,595,171,400,157,235,894,299,844,629,658,929,213,860,933,979,746,823,470,612,569,229,843,786,166,403,189,30,85,310,932,990,788,754,212,115,597,787,944,820,49,684,130,506,518,502,266,430,34,874,926,395,67,244,459,573,108,708,990,4,143,900,183,139,768,48,235,15,843,848,884,980,336,631,818,230,688,395,113,931,787,960,110,475,780,442,234,122,344,795,989,751,340,927,243,289,849,483,258,124,757,823,779,630,141,865,446,769,255,375,350,449,40,52,407,335,264,681,312,532,911,573,588,927,83,283,922,375,1,829,403,936,855,814,744,326,554,232,691,684,254,297,213,167,678,51,391,82,187,697,910,1,535,530,980,677,775,872,769,662,371,158,702,861,736,742,864,235,615,507,52,524,829,897,764,855,67,161,927,385,47,333,182,726,680,600,546,726,612,259,168,36,385,282,491,364,591,599,496,893,348,778,321,718,754,10,121,605,389,339,389,360,943,393,235,353,146,572,67,602,465,189,952,624,643,272,836,258,329,400,240,470,360,590,730,523,965,308,993,508,464,673,618,388,644,328,811,40,226,525,190,373,700,453,126,728,50,898,124,292,324,551,520,570,786,462,2449,887,642,697,289,51,556,589,956,935,782,331,499,579,595,72,21,114,369,432,730,576,767,869,223,776,128,667,594,41,493,219,913,642,728,341,481,313,170,567,544,104,21,739,341,312,472,563,771,833,517,471,231,20,536,766,693,28,468,733,973,821,470,963,659,202,687,194,319,388,422,6,250,767,115,797,367,465,545,310,582,160,935,435,673,869,304,421,575,993,535,392,114,755,313,881,547,479,98,61,796,709,756,810,248,296,582,184,151,856,988,9,871,80,897,957,267,908,807,876,969,442,636,72,685,680,415,162,559,925,551,254,891,960,947,179,900,28,633,891,528,854,91,369,747,751,285,458,948,436,892,735,287,233,59,464,163,614,910,170,777,435,261,40,259,780,191,292,92,834,586,182,288,968,307,235,140,50,648,427,964,41,476,540,918,227,879,327,153,812,232,862,650,127,238,312,266,940,921,804,482,158,786,756,874,492,495,548,851,911,315,383,175,940,492,5,380,521,260,945,526,224,180,456,136,854,37,57,934,82,115,923,114,511,691,556,375,935,833,866,338,146,422,737,476,727,291,701,614,732,391,415,762

Find the average and median of this set of 999 numbers  
(The median of a set of numbers is the number which has half of the other numbers less than it, and half greater than it)

File: **13b-numbers.txt**

# Example

Given a dictionary consisting of comma-separated words, make a webpage which performs spellchecks on words that the user enters, by seeing if they exist in the dictionary

“a,aah,aahed,aahing,aahs,aardvark,aardvarks,aard wolf,aardwolves,ab,aba,abac,abaca,abaci,aback,ab acs,abacus,abacuses,...” .. and 137000 more...

- See next slide

```
<html>
<head>
<script>
var dictionary;

function init() {
    var txt = "a,aah,aahed,aahing,aahs,aardvark,aardvarks,aardwolf"; // and a lot more!
    dictionary = txt.split(",");
    alert("There are "+dictionary.length+" words in the dictionary");
}

function checkUserWord() {
    var word = document.getElementById("txtUserWord").value;
    word = word.toLowerCase();

    for (var i=0; i<dictionary.length; i++) {
        if (word==dictionary[i]) {
            alert(word+" is in the dictionary!");
            return;
        }
    }
    alert(word+" was not found");
}
```

```
</script>
</head>
<body onload="init();">
<input type='text' id='txtUserWord'> <button
onclick='checkUserWord();'>Check Word</button>
</body>
</html>
```

# Discussion

- Is the approach taken to search the dictionary in the previous example efficient?
  - Linear search of 137000+ items
- How could we do it more efficiently?

# Exercise (not graded)

- Modify the previous program so that it performs the search more efficiently:
  - Binary search?
  - Hash table?
- How much faster is it?

# **CT1114**

# **Web Development**

## **Section 14:**

## **More CSS**

---

Dr. Sam Redfern

[sam.redfern@nuigalway.ie](mailto:sam.redfern@nuigalway.ie)

@psychicsoftware



NUI Galway  
OÉ Gaillimh

# The CSS Box Model

All HTML elements considered as boxes

Paragraphs, Headings, Tables, etc

Box Model

A box that wraps around all the HTML elements

Consists of Four Parts:

Content

Padding

Border

Margin

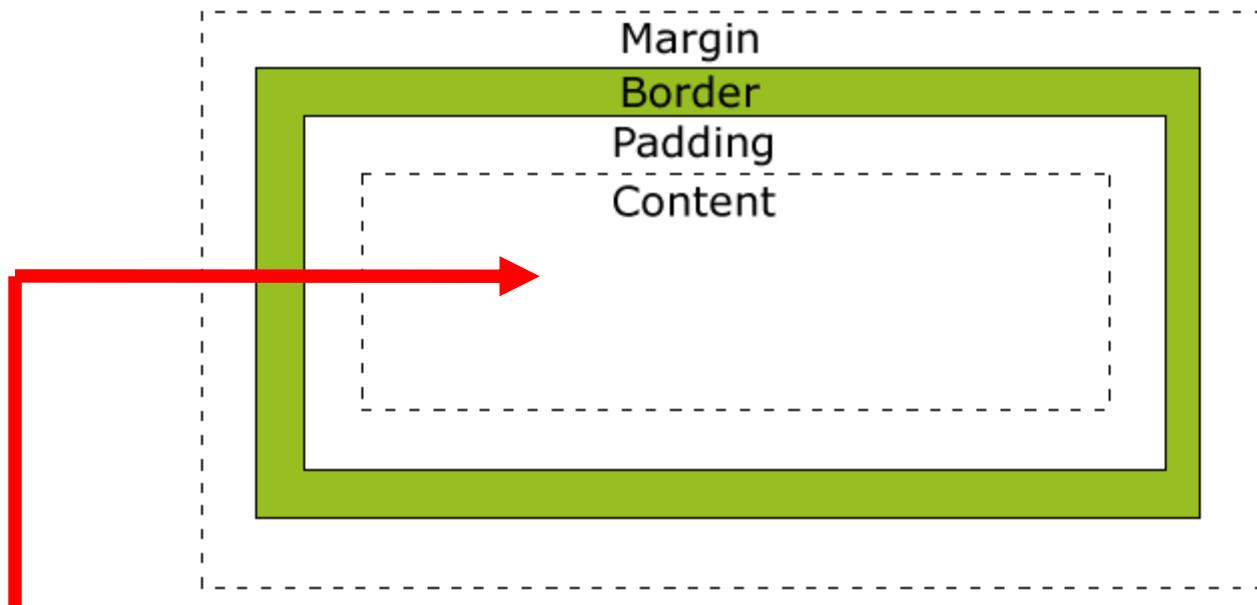
# CSS Box Model

- Content
- Padding
- Border
- Margin



- Padding, Border & Margin are ZERO by default**

# Content Area



## Content

- The HTML element concerned
- Text, Image, List, Table, etc

# Content – Embedded <p> Style Example

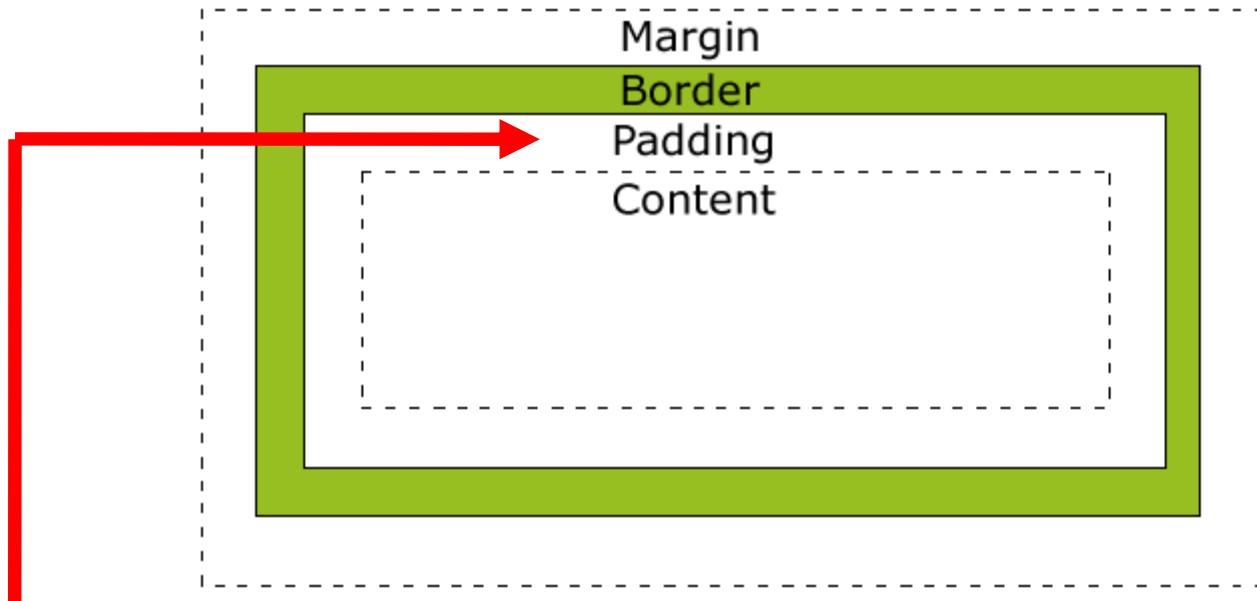
```
<html>
<head>
<style>
p
{
    background: #00FFFF;
}
</style>
</head>
```

```
<body>
<p>Text Goes Here!</p>
</body>
</html>
```

**Result:**

Text Goes Here!

# Padding Area



## Padding

- Empty space surrounding the Content
- Uses the **same** background colour as the Content

# Padding Content

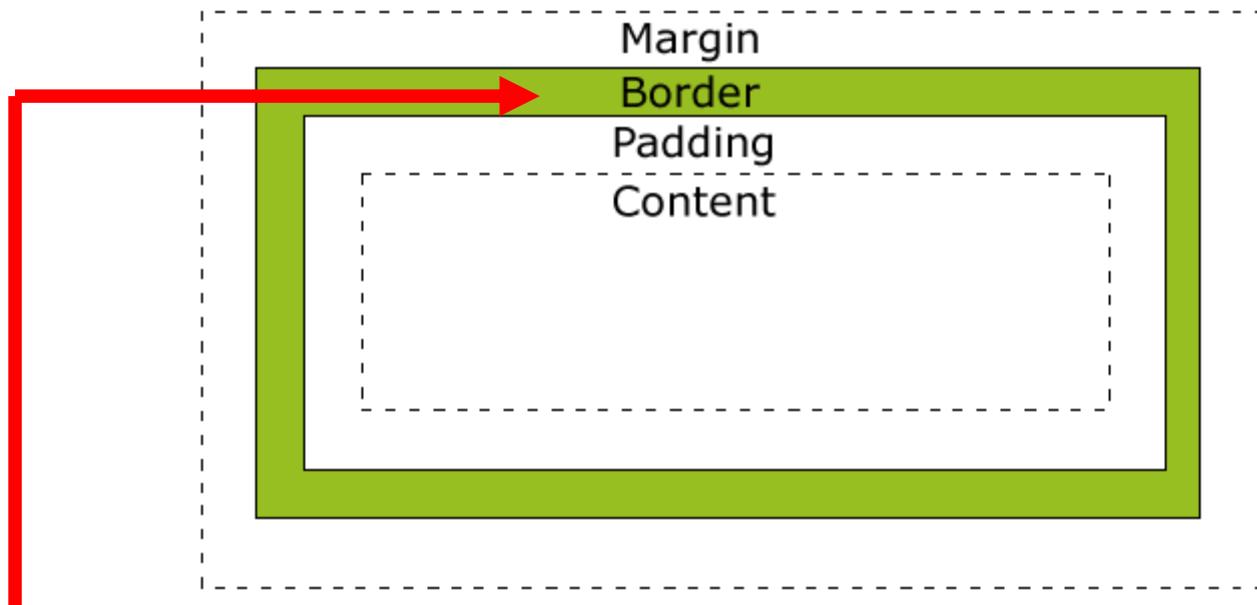
```
p  
{  
background: #00FFFF;  
padding: 0px;  
}
```

Text Goes Here!

```
p  
{  
background: #00FFFF;  
padding: 20px;  
}
```

Text Goes Here!

# Border Area



## Border

- The HTML element concerned
- `border-style` **must** be set for border to take effect

# Bordering Content

```
p  
{  
background: #00FFFF;  
padding: 20px;  
border: 20px;  
border-color: #FF0000;  
}
```

Text Goes Here!



Missing `border-style`,  
so no border displays

# Bordering Content

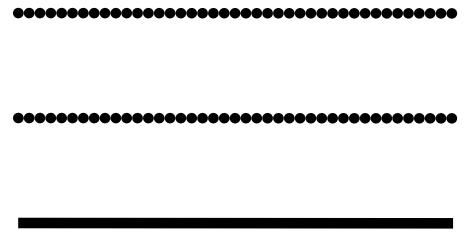
```
p  
{  
background: #00FFFF;  
padding: 20px;  
border: 20px;  
border-color: #FF0000;  
border-style: solid;  
}
```



# border-style values

none  
dotted  
dashed  
solid  
double

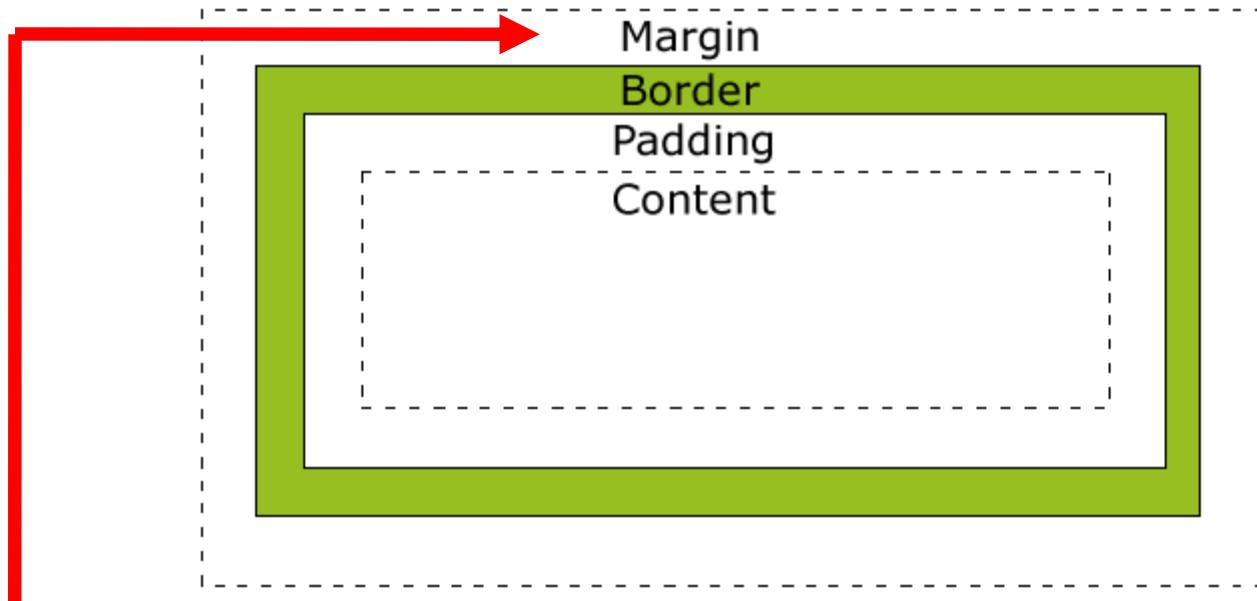
*No border*  
*Dotted border*  
*Dashed border*  
*Solid border*  
*Two solid borders*



groove  
ridge  
inset  
outset

*3D "grooved" border (engraved)*  
*3D "ridged" border (emboss)*  
*3D "inset" border (lowered)*  
*3D "outset" border (raised)*

# Margin Area



## Margin

- Transparent area that surrounds everything else
- Used for spacing the element relative to others

# Width and Height

Set through width and height properties

```
width: 100px;
```

```
height: 40px;
```

Affects **only** the Content of the Box Model

# Width Example

```
p  
{  
width: 100px;  
padding: 10px;  
border: 10px;  
border-style: solid;  
}
```

Total width:  
100 (Content)  
+ 20 (Padding L/R)  
+ 20 (Border L/R)  
= 140 pixels

# Side Independent Sizes

Individual sides can be given their own sizes

Left, Top, Right, Bottom

Done by adding the side to property

border-left: 10px;	<i>10 pixel left border</i>
border-top: 20px;	<i>20 pixel top border</i>
border-right: 5px;	<i>5 pixel right border</i>
border-bottom: 40px;	<i>40 pixel bottom border</i>

**Padding, Border or Margin work too**

e.g., padding-left or margin-top

# Side Independent Sizes – Shorthand

Possible to declare the sides in one declaration

border: 10px 20px 5px 40px;

## Order is important

Top, Right, Bottom, Left

*4 values declared*

Top, **Right/Left**, Bottom

*3 values declared*

Top/Bottom, Right/Left

*2 values declared*

All four take value

*1 value declared*

Also applies to border-style

border-top-style **or** border-right-style

```
<html>
<head>
<style>
span
{
    width: 100px;
    padding: 10px;
    border: 3px;
    border-top-style: solid;
    border-bottom-style: dotted;
    border-left-style: dashed;
    border-right-style: groove;
}
</style>
</head>
<body>
<p><br>
```

```
<span>
    Hey, it's a span
</span>
<span>
    Hey, it's another span
</span>
</p>
</body>
</html>
```

Hey, it's a span

Hey, it's another span

.....

# Grouping Selectors

- Can apply common style to more than one selector
- Do so by grouping the selectors together
- Example, centre aligning all Heading elements:

```
h1, h2, h3, h4, h5, h6
{
    text-align: center;
}
```

# What Takes Priority?

Consider:

```
h1, h2, h3, h4, h5, h6
{
  color: #FF0000;
}
```

**Local Element Style** will  
always take priority over a  
grouped style

```
h1
{
  color: #0000FF;
}
```

In this case, h1's colour will  
be **blue - #0000FF**

# Apply General Style

Can use Asterisk Wildcard ( \* ) as a selector to apply a general style to **all** elements

```
*  
{  
    font-family: Arial, sans-serif;  
}
```

As per previous slide, **local element styles** will over-write any general style for that element

# Style Classes

Allows for multiple elements to be styled with a single class

Useful for things like paragraphs

- Multiple text alignments

- Left, Center, Right, Justify

Classes created by naming selector .name

Classes are called by using the `class` attribute

Classes inherit the style of the **parent element**

**Unless overwritten by class** (attribute use only)

# Example

```
p  
{  
color: #0000FF;  
}
```

```
.left  
{  
text-align: left;  
}
```

```
.right  
{  
text-align: right;  
color: #FF0000;  
}
```

```
.center  
{  
text-align: center;  
}
```

```
.justify  
{  
text-align: justify;  
}
```

Elements using  
the "right" class  
will also be red

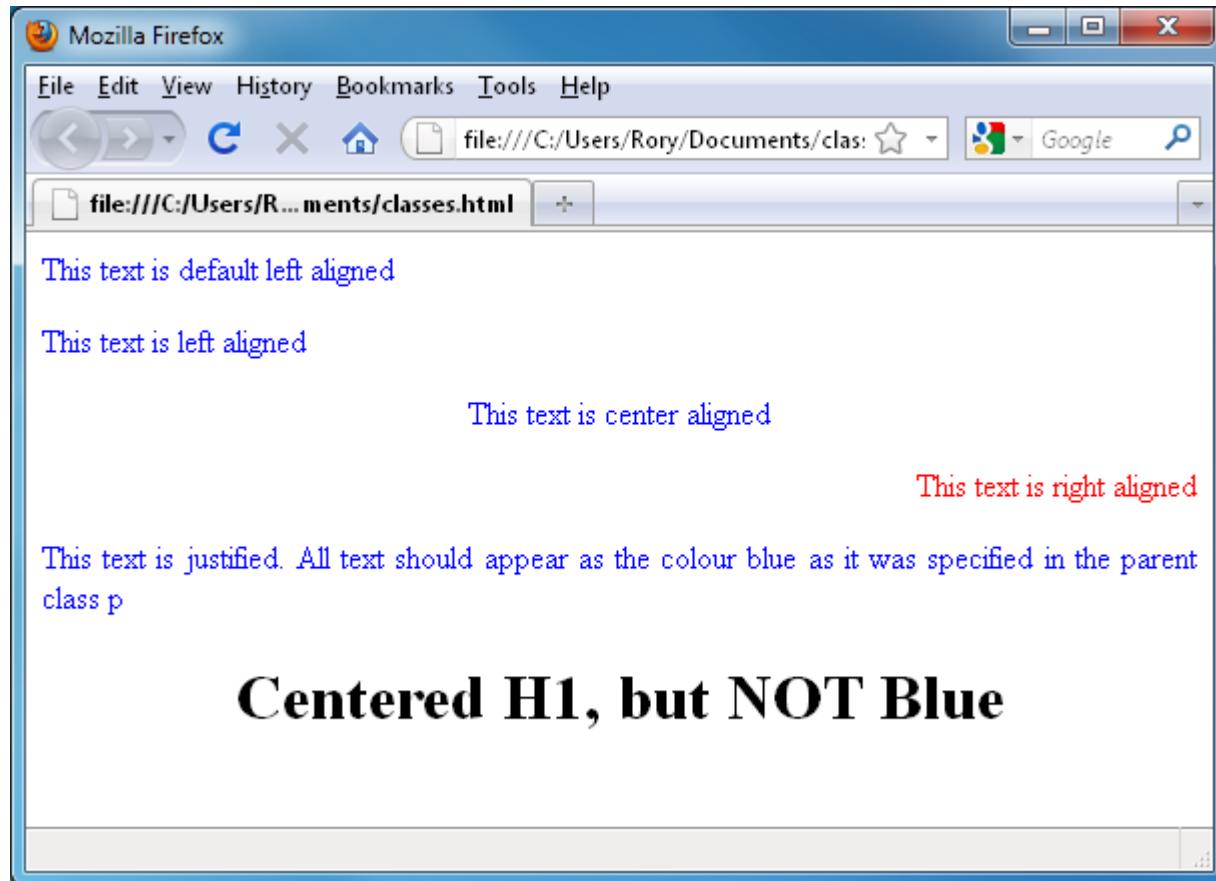
# Alignment Classes – Page Source

```
<p>This text is default left aligned</p>
<p class="left">This text is left aligned</p>
<p class="center">This text is center aligned</p>
<p class="right">This text is right aligned</p>
<p class="justify">This text is justified. All text
should appear as the colour blue as it was specified
in the parent paragraph class</p>

<h1 class="center">Centered H1, but NOT Blue</h1>
```

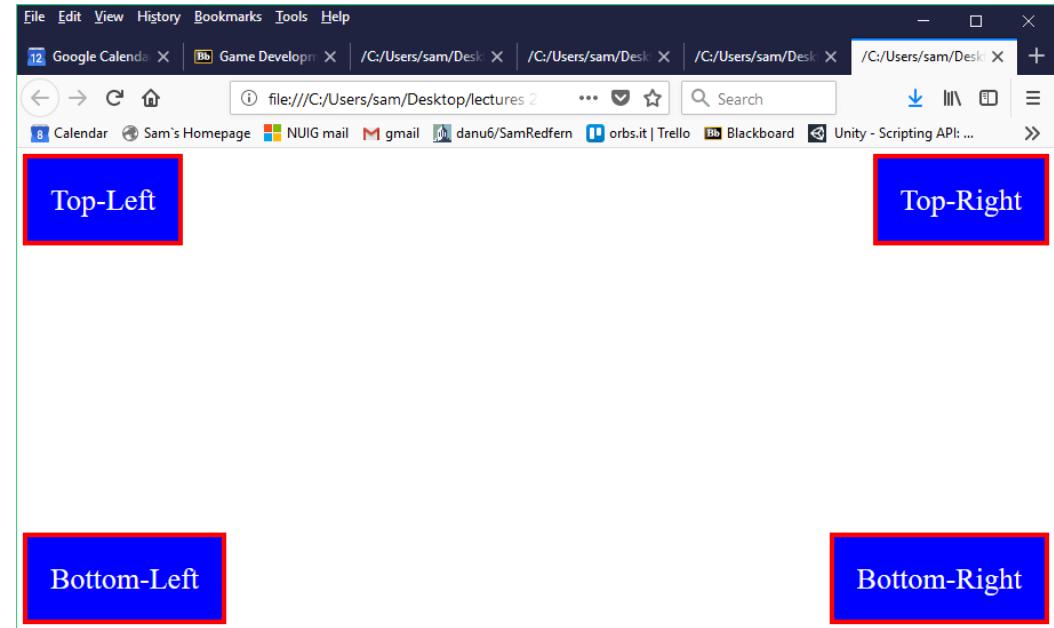
h1 is not blue as **only** the paragraph had blue colour

# Page Output



# Exercise

Write a CSS file  
(cornerstyles.css) which can  
be used by the following  
HTML document in order to  
produce the depicted output



```
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" type="text/css" href="cornerstyles.css" />
</head>
<body>
    <div class='topleft'>Top-Left</div>
    <div class='bottomleft'>Bottom-Left</div>
    <div class='topright'>Top-Right</div>
    <div class='bottomright'>Bottom-Right</div>
</body>
</html>
```

# Alternative units of size/distance in CSS: ‘em’

- In **CSS**, an **em** unit is equal to the computed font-size for the element to which the **em** is applied.
- If no font size is defined anywhere in the **CSS**, the **em** unit will be equal to the browser's default font size for the document, which is usually 16px.
- You can define the sizes of elements using em rather than px, in order to define them relative to each other and then perhaps easily change them all at once

# CSS ‘display’ attribute

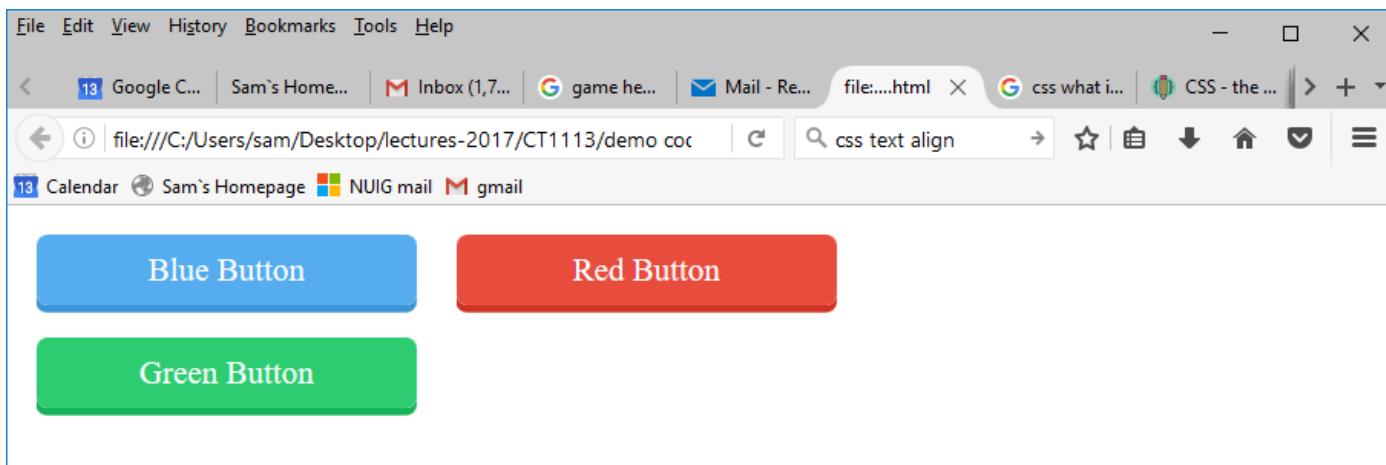
- The display attribute controls how an element is positioned
- `display:block;` the element starts on a new line and stretches out to the left and right as far as it can (default for DIV tags)
- `display:inline;` the element does not interrupt the flow of the paragraph, but still defines a section of content that you can control. (default for SPAN tags)
- `display:inline-block;` the same as inline, except the element is allowed to have a width and height
- `display:none;` the element is not displayed at all, and the page is unaffected by it, as if it doesn’t exist (useful for hiding content and revealing it later)

# Example: CSS buttons

```
html {  
    font-size:18px; /* we can dynamically change this  
based on screen size, and everything else is relative to  
this as em units */  
    color:white;  
}  
  
.btn {  
    border-radius: 0.333em;  
    padding: 0.5em 0.4167em;  
    font-size: 1.167em;  
    text-align: center;  
    text-decoration: none;  
    margin: 0.5em;  
    color: #ffffff;  
    position: relative;  
    display: inline-block;  
    min-width: 11em;  
}  
  
.btn:active {  
    transform: translate(0em, 0.25em);  
    -webkit-transform: translate(0em, 0.25em);  
    box-shadow: 0em 0.0833em 0em 0em;  
}  
  
.blue {  
    background-color: #55acee;  
    box-shadow: 0em 0.25em 0em 0em #3C93D5;  
}  
  
.blue:hover {  
    background-color: #6FC6FF;  
}  
  
.green {  
    background-color: #2ecc71;  
    box-shadow: 0em 0.25em 0em 0em #15B358;  
}  
  
.green:hover {  
    background-color: #48E68B;  
}  
  
.red {  
    background-color: #e74c3c;  
    box-shadow: 0em 0.25em 0em 0em #CE3323;  
}  
  
.red:hover {  
    background-color: #FF6656;  
}
```

# Example: CSS buttons

```
<html>
<head>
  <link rel="stylesheet" type="text/css" href="css_buttons.css" />
</head>
<body>
  <a class="btn blue">Blue Button</a> <a class="btn red">Red Button</a><br>
  <a class="btn green">Green Button</a>
</body>
</html>
```



# Graded Exercise

Sun	Mon	Tue	Wed	Thu	Fri	Sat
				1	2	3
4	5	6	7	8	9	10
11	<b>12</b>	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28			

- Write an HTML/Javascript page which requests from the user:
  - The number of days in the month
  - The day of the week (0-6) that the 1<sup>st</sup> of the month falls on
- Your Javascript code should then construct a calendar similar to the one shown above, from the supplied information
- The calendar should be structured using a <table> and displayed in a <div>. You should use css styling to colour it.
- Note that the pink table cell indicates the current day of the month, obtained from a Date() object

# CT1114

## Web Development

### Section 15: Calendar Calculations using Javascript

Dr. Sam Redfern

[sam.redfern@nuigalway.ie](mailto:sam.redfern@nuigalway.ie)

@psychicsoftware



NUI Galway  
OÉ Gaillimh

# Last Week's Graded Exercise

Sun	Mon	Tue	Wed	Thu	Fri	Sat
				1	2	3
4	5	6	7	8	9	10
11	<b>12</b>	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28			

- Write an HTML/Javascript page which requests from the user:
  - The number of days in the month
  - The day of the week (0-6) that the 1<sup>st</sup> of the month falls on
- Your Javascript code should then construct a calendar similar to the one shown above, from the supplied information
- The calendar should be structured using a <table> and displayed in a <div>. You should use css styling to colour it.
- Note that the pink table cell indicates the current day of the month, obtained from a Date() object

# Drawing the calendar: algorithmic steps

- Using a loop, create correct number of grey-coloured table cells for ‘missing’ days in 1<sup>st</sup> week
- Using another loop, create a blue-coloured table cell for each day in the month. Treat Saturday each week as a special case which requires a closing row tag </TR> followed by an opening row tag <TR> for the next line
- Using a third loop, draw any remaining grey cells to bring the final week up to Saturday
- Each loop follows the previous one, i.e. \*not\* nested!
- Optionally, we can also colour a cell a special colour if it is today (according to system clock)

# Making a complete calendar webpage

Sun	Mon	Tue	Wed	Thu	Fri	Sat
				1	2	3
4	5	6	7	8	9	10
11	<u>12</u>	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28			

- Previously, we asked the user for:
  - The number of days in the month
  - The day of the week (0-6) that the 1<sup>st</sup> of the month falls on
- But surely, given a month number and year number, we can calculate these things using Javascript?

# How many days does our month have?

```
function no_of_days(year, month) {  
    // Sept,Apr,Jun,Nov  
    if (month == 9 || month == 4 || month == 6 || month == 11)  
        return 30;  
  
    // All others except Feb  
    if (month != 2)  
        return 31;  
  
    // but what about Feb.. is it 28 or 29?  
    // ??  
}
```

# Leap Years

- What are the rules for determining if a year is a leap year?
  1. ?
  2. ?
  3. ?

# Exercise: Leap Years

- Write a Javascript function which receives a year number as an argument.
- The function should return 1 if the year is a leap year, and return 0 if it is not.
- Start with the code provided on the next slide

# Exercise: Leap Years

```
<!DOCTYPE html>
<html>
<head>
<script>
    function is_leap(year) {
        // write your code here!
        // return 1 if year is a leap year
        // return 0 if year is not a leap year
    }

    function checkYear() {
        var txt = document.getElementById("txtYear").value;
        var year = parseInt(txt);
        if (isNaN(year))
            alert(txt + " is not a number!");
        else if (is_leap(year)==1)
            alert(year + " is a leap year");
        else
            alert(year + " is not a leap year");
    }
</script>
</head>
<body>
    <input id='txtYear'> <button onclick='checkYear();'>Check Year</button>
</body>
</html>
```

# How many days does our month have?

```
function no_of_days(year, month) {  
    // Sept,Apr,Jun,Nov  
    if (month == 9 || month == 4 || month == 6 || month == 11)  
        return 30;  
  
    // All others except Feb  
    if (month != 2)  
        return 31;  
  
    // Feb  
    return (28+is_leap(year));  
}
```

This function now correctly deals with leap years

# What day of the week does our month start on?

- Baseline fact: January 1<sup>st</sup>, 1900 was a Monday
- So, what day of the week was February 1<sup>st</sup>, 1900?
- And how do we calculate that?

# Generalising this rule

- The day of the week that the 1<sup>st</sup> of any month (after Jan 1900) falls on is:
- $(1 + \text{the number of days that have elapsed since Jan } 1^{\text{st}}, 1900) \% 7$
- Question: what day of the week was May 1<sup>st</sup>, 1900?
  - Note that 1900 was not a leap year

# What about years later than 1900?

- We can calculate that Jan 1<sup>st</sup> 1905 was a Sunday..
- How.. ?

# This Week's Graded Exercise

- Make an HTML page which lets the user enter a year number and a month number
- The page should then format and display a calendar for that month and year (using last week's assignment as a basis)
  - You can use my solution to last week's static calendar to start with, or else your own
- Your solution should use functions such as those we have been discussing, i.e. don't simply manipulate Date objects to find out information such as days of the week

# CT1114

# Web Development

## Section 16:

## HTML (Forms), Javascript

Dr. Sam Redfern

[sam.redfern@nuigalway.ie](mailto:sam.redfern@nuigalway.ie)

@psychicsoftware



NUI Galway  
OÉ Gaillimh

# HTML Forms

- Forms provide a standard set of input controls: text fields, buttons, check boxes, lists, etc.
- The user's choices are entered into the controls
- The contents of the controls can be manipulated and processed on the client (using Javascript), prior to posting them to the webserver
- or (after being posted) they can be processed on the server, using a server-side language such as PHP.
- Any arbitrary programming functions can be performed with the posted data, once the server has it: Calculations, Database entries, Email, Credit Card charges, etc.
- To post the form to the server, either:
  - Provide an `<input type='submit'>` field in the form
  - Or, use the form object's `.submit()` method in Javascript

# <form>

Attributes of the <form> container tag:

ACTION            defines the URL that the data will be posted to

METHOD defines how the data is sent:

    POST – the form element names and values are sent as a ‘Request’ object which the server-side script can read

    GET – the form element names and values are sent as part of the URL using ‘query strings’

Form controls:

<textarea> defines a field in which the user can type multiple lines of text.

<select> enables the user to choose among a number of options in a list

<input> provides all of the other types of input: single lines of text, radio buttons, check boxes

<button> push-buttons

The next few slides illustrate form controls in HTML

# Form Layout

```
<form method="post" action="script.php">  
    <!-- form elements such as text fields are added here -->  
</form>
```

method

How the field values will be processed (either ‘post’ or ‘get’)

action

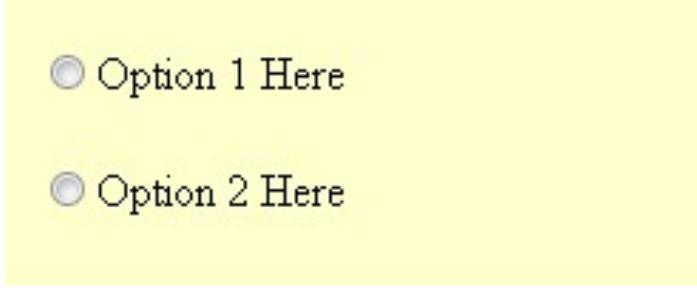
The server-side script which the field values will be passed to

# <form> elements

|                         |                              |
|-------------------------|------------------------------|
| <select>                | drop-down lists              |
| <option>                | items in <select> lists      |
| <input type='checkbox'> | checkboxes                   |
| <input type='radio'>    | radio buttons                |
| <input>                 | single-line text boxes       |
| <textarea>              | multi-line text boxes        |
| <input type='submit'>   | 'submit form' push-buttons   |
| <button>                | general purpose push-buttons |

# Multiple Choice – Radio Buttons

```
<input type="radio" name="grouplabel"  
value="1">Option 1 Here  
<br><br>  
<input type="radio" name="grouplabel"  
value="2">Option 2 Here
```



Option 1 Here

Option 2 Here

User can only select **one** option with radios  
(identified as a group by **name**)

# Multiple Choice – Check Boxes

```
<input type="checkbox" name="label"  
checked="checked">Option 1 Here  
<br><br>  
<input type="checkbox" name="label2">Option 2 Here
```

The image shows a yellow rectangular box containing two lines of text. The first line contains a checked checkbox followed by the text "Option 1 Here". The second line contains an unchecked checkbox followed by the text "Option 2 Here".

Option 1 Here

Option 2 Here

# Multiple Choice – Drop Down List

```
<select name="listlabel" id="listid">
  <option value="1">Item 1</option>
  <option value="2" selected="selected">Item
2</option>
  <option value="3">Item 3</option>
</select>
```



To find the 0-based index of the selected item at run-time, using Javascript:

```
var idx = document.getElementById("listid").selectedIndex;
```

# Exercise

- Make an HTML page which contains a `<select>` and a `<button>`.
- The `<select>` should have 12 `<option>`s, containing the 12 months of the year
- When the user clicks the button, use an alert to tell them which month number they selected (1 - 12)

# Example: submitting a form to the server following client-side validation

```
<html>
<head>
<script>
function validateForm() {
    var a = document.getElementById("field1").value;
    var b = document.getElementById("field2").value;
    if (a<1 || a>10)
        alert("The first number is not between 1 and 10");
    else if (b<1 || b>10)
        alert("The second number is not between 1 and 10");
    else {
        alert("All ok.. will submit the form now..");
        document.getElementById("theForm").submit();
    }
}
</script>
</head>
<body>
<form id='theForm' method='post' action='page.php'>
    Please enter two numbers, each between 1 and 10<br>
    <input name='field1' id='field1'><br>
    <input name='field2' id='field2'><br>
    <button onclick='validateForm();'>Submit</button>
</form>
</body>
</html>
```

# Graded Exercise

Question Category	<input type="select" value="Goods for Sale"/> Goods for Sale
Question	<input type="text" value="Do you have any widgets in stock?"/>
Your Name	<input type="text" value="Sam"/>
Your Email	<input type="text" value="yosemite@acme.com"/>
	<input type="button" value="Submit"/>

The HTML form depicted is used to collect information from customers on a website. It has been laid out on the page through use of a <table>. It contains:

- a drop-down <select> control containing the options: “Goods for Sale”, “Complaints”, and “Support”
- a <textarea> into which the user types a question
- two text <input> boxes, into which the user enters their name and email address
- a button labelled ‘Submit’

Write suitable HTML+CSS code to produce this form. Write a Javascript function to check the validity of the form, when the Submit button is clicked. To be valid, each of the fields must contain some text, and the text in the Email field must include an '@' symbol and a '.' symbol.

# CT1114

# Web Development

## Section 16:

## HTML (Forms), Javascript

Dr. Sam Redfern

[sam.redfern@nuigalway.ie](mailto:sam.redfern@nuigalway.ie)

@psychicsoftware



NUI Galway  
OÉ Gaillimh

# HTML Forms

- Forms provide a standard set of input controls: text fields, buttons, check boxes, lists, etc.
- The user's choices are entered into the controls
- The contents of the controls can be manipulated and processed on the client (using Javascript), prior to posting them to the webserver
- or (after being posted) they can be processed on the server, using a server-side language such as PHP.
- Any arbitrary programming functions can be performed with the posted data, once the server has it: Calculations, Database entries, Email, Credit Card charges, etc.
- To post the form to the server, either:
  - Provide an `<input type='submit'>` field in the form
  - Or, use the form object's `.submit()` method in Javascript

# <form>

Attributes of the <form> container tag:

ACTION            defines the URL that the data will be posted to

METHOD defines how the data is sent:

    POST – the form element names and values are sent as a ‘Request’ object which the server-side script can read

    GET – the form element names and values are sent as part of the URL using ‘query strings’

Form controls:

<textarea> defines a field in which the user can type multiple lines of text.

<select> enables the user to choose among a number of options in a list

<input> provides all of the other types of input: single lines of text, radio buttons, check boxes

<button> push-buttons

The next few slides illustrate form controls in HTML

# Form Layout

```
<form method="post" action="script.php">  
    <!-- form elements such as text fields are added here -->  
</form>
```

method

How the field values will be processed (either ‘post’ or ‘get’)

action

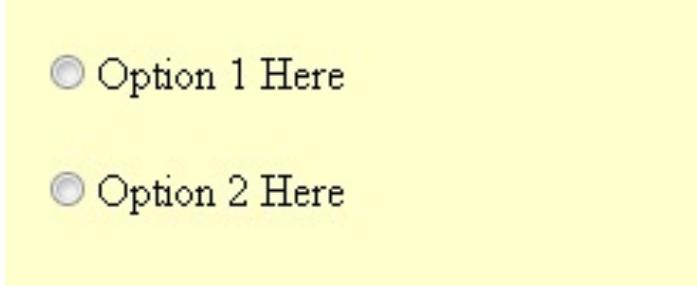
The server-side script which the field values will be passed to

# <form> elements

|                         |                              |
|-------------------------|------------------------------|
| <select>                | drop-down lists              |
| <option>                | items in <select> lists      |
| <input type='checkbox'> | checkboxes                   |
| <input type='radio'>    | radio buttons                |
| <input>                 | single-line text boxes       |
| <textarea>              | multi-line text boxes        |
| <input type='submit'>   | 'submit form' push-buttons   |
| <button>                | general purpose push-buttons |

# Multiple Choice – Radio Buttons

```
<input type="radio" name="grouplabel"  
value="1">Option 1 Here  
<br><br>  
<input type="radio" name="grouplabel"  
value="2">Option 2 Here
```



Option 1 Here

Option 2 Here

User can only select **one** option with radios  
(identified as a group by **name**)

# Multiple Choice – Check Boxes

```
<input type="checkbox" name="label"  
checked="checked">Option 1 Here  
<br><br>  
<input type="checkbox" name="label2">Option 2 Here
```

The image shows a yellow rectangular box containing two lines of text. The first line has a checked checkbox followed by the text "Option 1 Here". The second line has an unchecked checkbox followed by the text "Option 2 Here".

[] Option 1 Here

[] Option 2 Here

# Multiple Choice – Drop Down List

```
<select name="listlabel" id="listid">
  <option value="1">Item 1</option>
  <option value="2" selected="selected">Item
2</option>
  <option value="3">Item 3</option>
</select>
```



To find the 0-based index of the selected option at run-time, using Javascript:

```
var idx = document.getElementById("listid").selectedIndex;
```

To find the value of the selected option, use the `options` array of the `select` object:

```
var s = document.getElementById("listid");
var val = s.options[s.selectedIndex].value;
```

# Exercise

- Make an HTML page which contains a `<select>` and a `<button>`.
- The `<select>` should have 12 `<option>`s, containing the 12 months of the year
- When the user clicks the button, use an alert to tell them which month number they selected (1 - 12)

# Example: submitting a form to the server following client-side validation

```
<html>
<head>
<script>
function validateForm() {
    var a = document.getElementById("field1").value;
    var b = document.getElementById("field2").value;
    if (a<1 || a>10)
        alert("The first number is not between 1 and 10");
    else if (b<1 || b>10)
        alert("The second number is not between 1 and 10");
    else {
        alert("All ok.. will submit the form now..");
        document.getElementById("theForm").submit();
    }
}
</script>
</head>
<body>
<form id='theForm' method='post' action='page.php'>
    Please enter two numbers, each between 1 and 10<br>
    <input name='field1' id='field1'><br>
    <input name='field2' id='field2'><br>
    <button onclick='validateForm();'>Submit</button>
</form>
</body>
</html>
```

# Graded Exercise

Question Category	<input type="select" value="Goods for Sale"/> Goods for Sale
Question	<input type="text" value="Do you have any widgets in stock?"/>
Your Name	<input type="text" value="Sam"/>
Your Email	<input type="text" value="yosemite@acme.com"/>
	<input type="button" value="Submit"/>

The HTML form depicted is used to collect information from customers on a website. It has been laid out on the page through use of a <table>. It contains:

- a drop-down <select> control containing the options: “Goods for Sale”, “Complaints”, and “Support”
- a <textarea> into which the user types a question
- two text <input> boxes, into which the user enters their name and email address
- a button labelled ‘Submit’

Write suitable HTML+CSS code to produce this form. Write a Javascript function to check the validity of the form, when the Submit button is clicked. To be valid, each of the fields must contain some text, and the text in the Email field must include an '@' symbol and a '.' symbol.

# **CT1114**

# **Web Development**

# **HTML, CSS, JavaScript**

## **Section 17:**

## **Ajax**

Dr. Sam Redfern

[sam.redfern@nuigalway.ie](mailto:sam.redfern@nuigalway.ie)

@psychicsoftware



NUI Galway  
OÉ Gaillimh

# Last Week's Graded Exercise

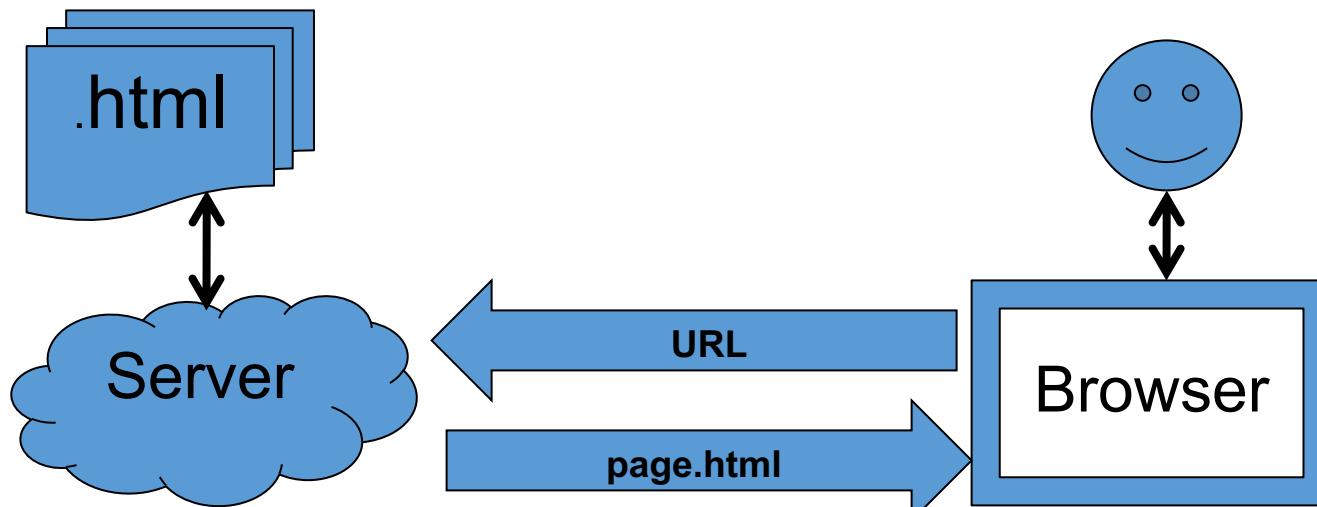
Question Category	<input type="select" value="Goods for Sale"/> Goods for Sale
Question	<input type="text" value="Do you have any widgets in stock?"/>
Your Name	<input type="text" value="Sam"/>
Your Email	<input type="text" value="yosemite@acme.com"/>
	<input type="button" value="Submit"/>

The HTML form depicted is used to collect information from customers on a website. It has been laid out on the page through use of a <table>. It contains:

- a drop-down <select> control containing the options: “Goods for Sale”, “Complaints”, and “Support”
- a <textarea> into which the user types a question
- two text <input> boxes, into which the user enters their name and email address
- a button labelled ‘Submit’

Write suitable HTML+CSS code to produce this form. Write a Javascript function to check the validity of the form, when the Submit button is clicked. To be valid, each of the fields must contain some text, and the text in the Email field must include an '@' symbol and a '.' symbol.

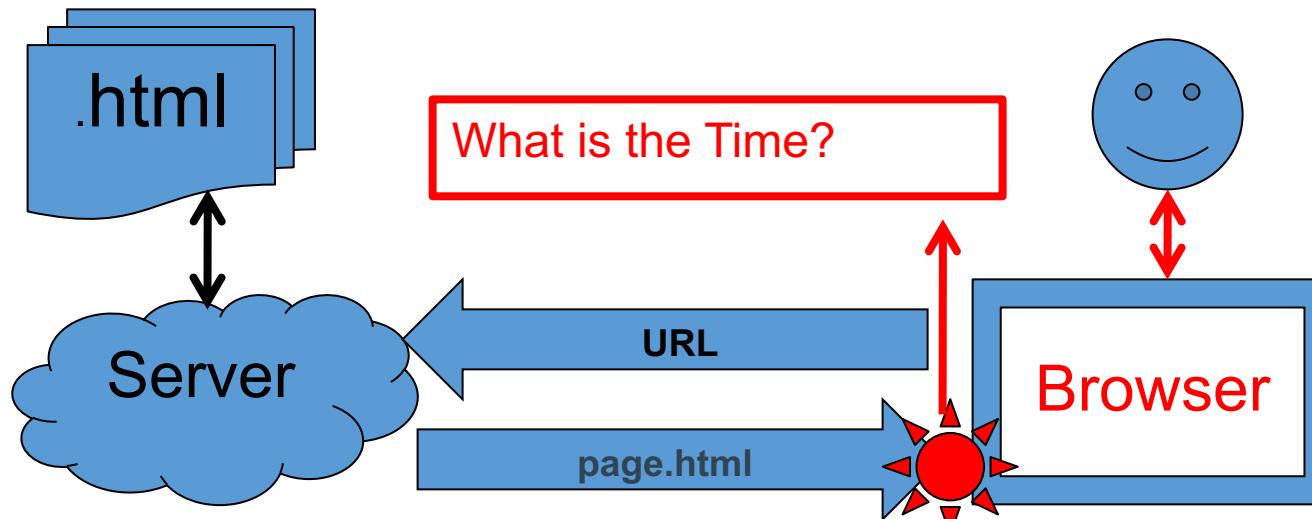
# Normal (Basic) HTML Interaction



1. Web Server stores HTML files
2. Client browser requests page through URL
3. Server sends HTML page to client
4. Client browser displays HTML page to User

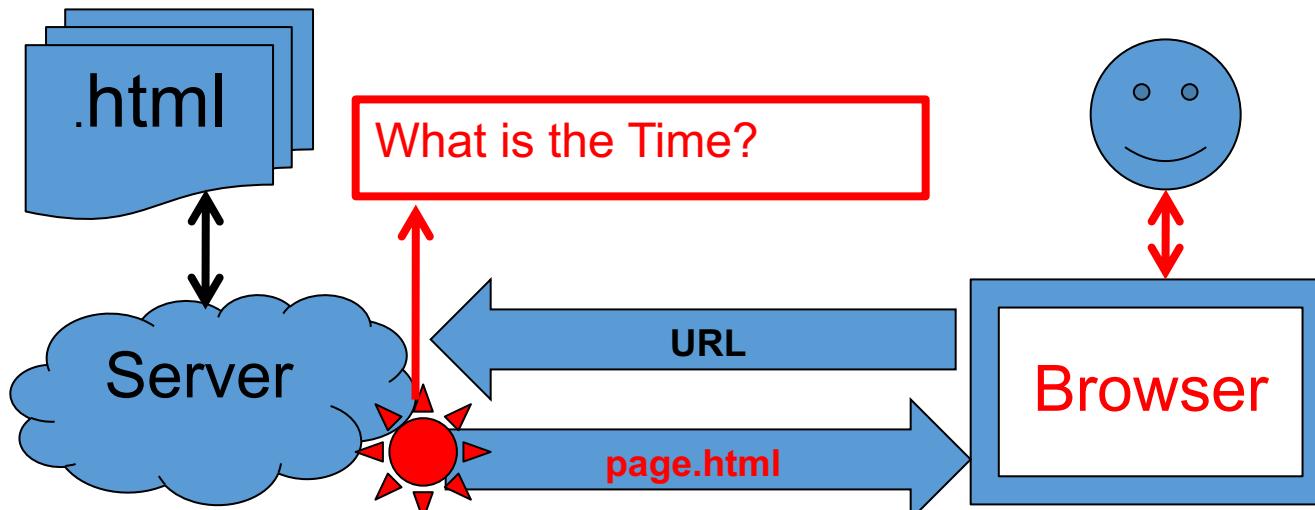
(Note that we haven't been using a server at all, we've been opening a local file directly into the browser. Notice the protocol in the address bar will be file:// instead of the normal http://)

# Client Side Scripting Interaction (using Javascript in the browser)



1. Web Server stores **HTML/Script** pages
2. Client browser requests page through URL
3. Server sends **HTML/Script** page to client
- 4. Client browser** runs script locally and updates HTML page
5. Client browser displays HTML page to User

# Server Side Scripting Interaction (using e.g. PHP or Node.js)



1. Web Server stores **HTML/Script** pages
2. Client browser requests page through URL
3. **Server runs script locally and updates HTML page**
4. Server sends **updated HTML** page to client
5. Client browser displays HTML page to User

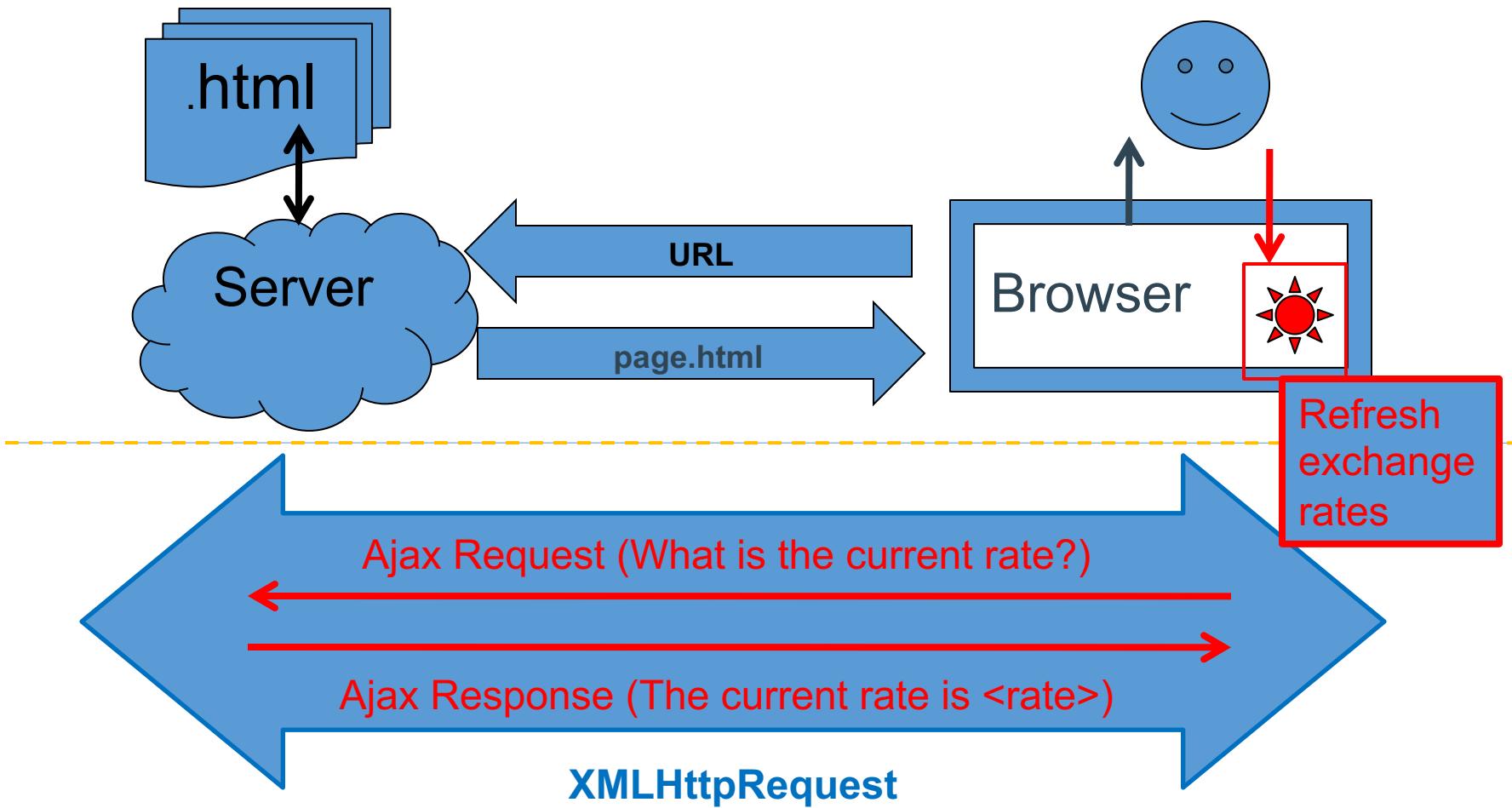
# Ajax – "Asynchronous Javascript and XML"

- A key component of modern web-apps
- Normal HTTP interactions instruct the browser to request an entire webpage at once
- Ajax, however, instructs the browser to request small pieces of content
  - Typically, smaller parts of the webpage
  - But can be any arbitrary data, not necessarily marked up as HTML
  - Asynchronous update => no tangible ‘refresh’ process for user
  - Often updated to individual elements on the page, e.g. via their .innerHTML attribute

# Requests that are not entire pages?

- A normal HTTP request to the server causes the entire page to be refreshed
  - The page is re-sent by the server and updated in the client's browser
  - All existing Javascript data on the page is destroyed
- Ajax requests **do not** cause the page to refresh
  - Only the appropriate content is updated (by JavaScript manipulating the existing Document)
  - e.g. Google Search Suggestions
- These requests are handled by the client separately to the normal HTTP requests
- On the server, Ajax requests are identical to normal HTTP requests

# Ajax Requests



# What is Ajax?

- **Asynchronous**
  - Request is made without interrupting the page
  - Doesn't happen at the same time as other code
  - Often user initiated by events, sometimes timed
- **JavaScript**
  - Asynchronously called-back whenever the Response has been received by the browser
- **XML**
  - Language agnostic mechanism for data transport and storage

# eXtensible Markup Language (XML)

- Designed to transport and store data in a human-readable yet structured format
  - Typically a simple text file of extension **.xml**
- Like HTML:
  - It is a semantic markup language made up of tags
    - i.e. It describes information
- Unlike HTML:
  - There are no standard tags
  - The tags are made up to fit the data being described
    - Tags are self-descriptive
- XML is increasingly being superseded by JSON (JavaScript Object Notation) as a standard data interchange format..  
see next week

# XMLHttpRequest

- Browser feature that allows JavaScript to make an Ajax request to the server without going through normal HTTP page-request process
  - i.e. the request is sent in the background without the page reloading
- XMLHttpRequest is an **Object** that must be **created** before it can be used:

```
var requester = new XMLHttpRequest();
```
- Once created we're ready to open a connection and send a request

# Opening an Ajax Connection

- Must first open a connection to the server
  - Connection is created via the **open()** method  
*requester.open (method, location,  
asynchronous);*
- Open function takes three arguments
  - Method:        GET or POST
  - Location:      URL to access
  - Asynchronous (true/false)?
    - Async: Allows other scripts to execute while request is going on
    - Sync:   Freezes the browser until request is done

# Sending the Request

- Once the connection is opened, the request can be sent via the **send()** method
- Example:

```
// Create the object  
var requester = new XMLHttpRequest();  
  
// Open the connection  
requester.open("POST", "page.php", true);  
  
// Send the request  
requester.send();
```

# GET or POST?

- Either can be used to pass info to the PHP page
- Using GET, add name=value pairs at end of location's URL, i.e. as a querystring
  - `requester.open("GET", "page.php?id=101&name=John", true);`
- Using POST, add an extra header value and send the information via the **send()** method
  - `requester.open("POST", "page.php", true);`
  - `requester.setRequestHeader("Content-type", "application/x-www-form-urlencoded");`
  - `requester.send("id=101&name=John");`

# Receiving the Response

- Use an Event Handler to execute when the server has returned a response
- Monitor for changes in the **readyState** property of the XMLHttpRequest object
  - Property has five states:
    - 0      uninitialized
    - 1      loading
    - 2      loaded
    - 3      interactive
    - 4      complete

# Change in readyState

- Monitored by the **onreadystatechange** event
- This event is triggered when the **readyState** changes
- The event should be handled by a function which is called when the state changes

```
requester.onreadystatechange = function()  
{  
    // code here  
}
```

# Change in readyState

```
requester.onreadystatechange = function()
{
    if (requester.readyState == 4 &&
        requester.status == 200)
    {
        // Then response has been received and
        // is okay.. So process data here!
    }
}
```

- Status code 200 means the request was okay

# Dealing with the Response

- There are two properties of XMLHttpRequest for handling the response
  - **responseXML** If the response is XML
  - **responseText** If the response is text or HTML
- In most cases, the response will be Text/HTML despite the fact that XML is part of the name Ajax
  - XML is not used so much these days; often we might send JSON data instead, formatted as a string in responseText
- The text can then be processed as required on the browser using Javascript
- For example it could be added to your page:

```
document.getElementById("mydiv").innerHTML = requester.responseText;
```

# Putting it All Together

- Assume we have a PHP script called **time.php**
- The time.php page will return the current server time when it is executed
- We have a **div** on our page to display the time in the correct place

```
<div id="divTime"></div>
```

- We want to send an Ajax request to update the time in this div

## Page with AJAX Function and Time Update Button [time.html](#)

```
<html>
  <head>
    <!-- see next slide for gettime.js -->
    <script src="gettime.js"></script>
  </head>
  <body>

    <h1>The time on the server is:</h1>
    <div id="divTime"></div>

    <input type="button" value="Update Time" onclick="getTime();">
  </body>
</html>
```

## gettime.js

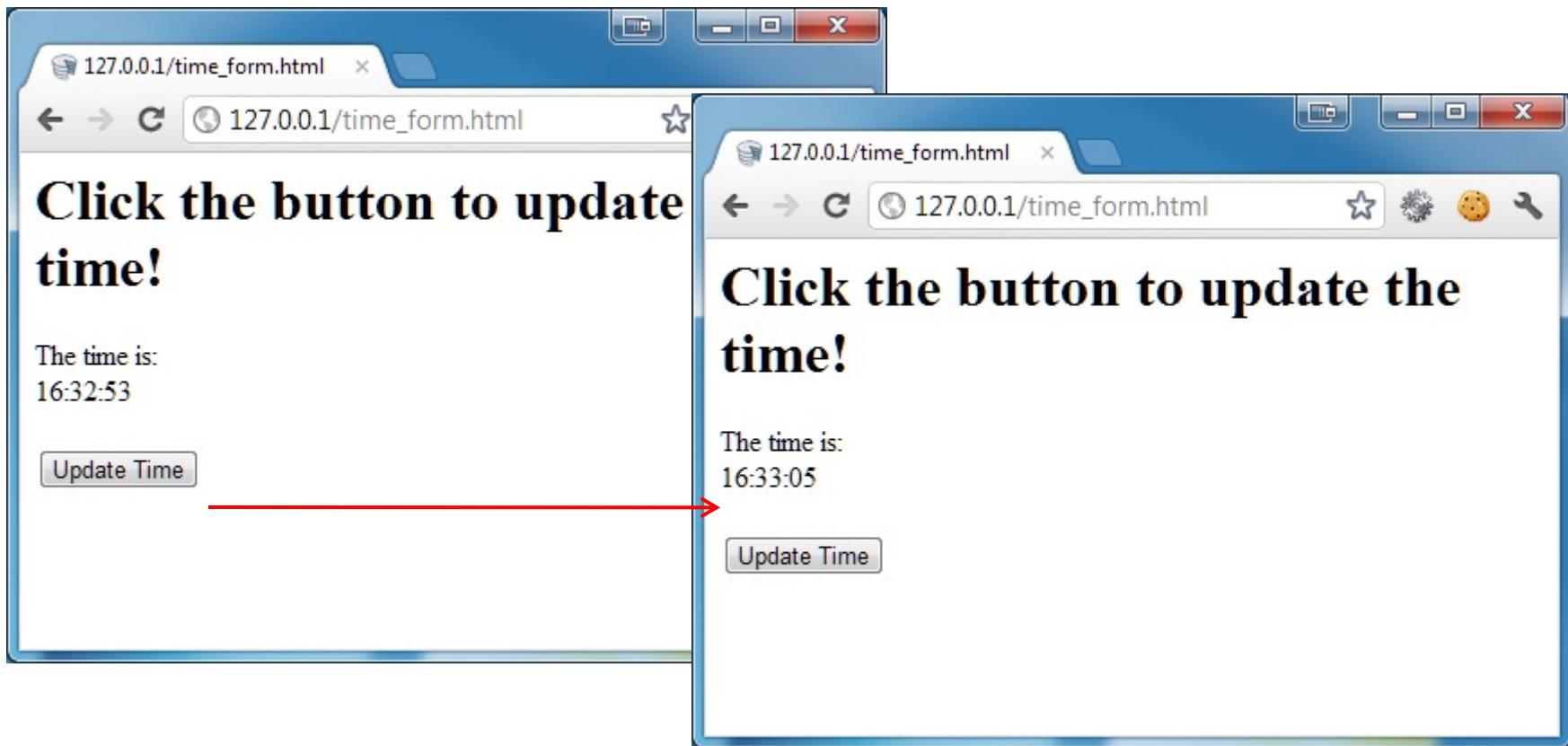
```
function getTime() {  
    // Create new XMLHttpRequest Object  
    var requester = new XMLHttpRequest();  
  
    /* Monitor the onreadystatechange event for a change in the readyState.  
    This function will be executed whenever the readyState is changed so must  
    be declared before the request is sent */  
    requester.onreadystatechange = function() {  
        if (requester.readyState == 4 && requester.status == 200) {  
            // Put the response into the div  
            document.getElementById("divTime").innerHTML =  
                requester.responseText;  
        }  
    }  
  
    // Open connection to time.php and send the request asynchronously  
    requester.open("GET", "https://www.dark-wind.com/ct1114time.php", true);  
    requester.send();  
}
```

time.php - this runs on a webserver, e.g. Apache

<?php

```
// Echo the current time in the format hh:mm:ss  
echo date('H:i:s');
```

?>



Q: Why does this PHP script echo just one thing, and not include <HTML> page tags etc.?

# Sequence of Last Example

- A button-click event was captured ***onclick()***
- Executed a JavaScript function ***getTime()***
- Sent an Ajax Request to a PHP page (executed on the server) ***time.php***
- Current time was echo'd into Response ***time.php***
- Ajax Response was received by Javascript ***readyState***
- The time was read from Response ***responseText***
- "divTime" div was updated ***innerHTML***

# Exercise

- Try out the last example! (using time.html and gettime.js)
- We don't have scope in CT1114 to study PHP code or using webservers, so instead I have made available some PHP scripts at specific URLs, so you can still practice Ajax from the client-side perspective
- ***The URL on which the ‘time’ php script is available is:***
- ***<https://www.dark-wind.com/ct1114time.php>***

# AJAX, PHP, Node.js, MySQL, WebSockets..

- The previous (server-time) example was a very simple example of Ajax
- More complex/realistic uses: e.g. query a database and update a page via Ajax
  - All without causing the full page to refresh
  - Useful e.g. if you have continuously-changing data to display on a webpage and don't want an ugly refresh process or the trouble it creates technically (e.g. JavaScript variables destroyed by page refresh)
- Modern 'single page' web-apps often use Ajax for server interactions, and are structured as single HTML/Javascript pages which never refresh
  - Sections of the app can be written into separate <div> tags, which are shown/hidden as appropriate
- An alternative to the Ajax approach is WebSockets which allow persistent (stateful) bi-directional communication channels to be established between a server and a browser.
  - WebSockets use TCP communication but don't use the HTTP: protocol.. Node.js is a very suitable server technology for WebSockets, but is beyond the scope of CT1114.
  - Websockets are often used for web-based chatrooms or even online multiplayer games (e.g. Agar.io, Orbs.it, Newbychinese.com)



# Exercise (not graded)

- Write a webpage which displays, alongside each other:
  - The time according to the user's browser
  - The time according to the server
  - The difference (in seconds) between these
- You should use Ajax to periodically retrieve the server's time (every 0.5 seconds) before updating the webpage with the above values
  - Question: why not just do it every 1.0 seconds?
- Hint: recall the use of **setTimeout** to repeatedly (periodically) call a Javascript function
- Question: how can we calculate the difference in seconds between the server and browser's reported time?
- **As before, you can access `ct1114time.php` here:**
- <https://www.dark-wind.com/ct1114time.php>

# CT1114

# Web Development

# HTML, CSS, JavaScript

## Section 18:

## Javascript Objects; Exploring Callbacks;

## ES6 Syntax; Revision

Dr. Sam Redfern

[sam.redfern@nuigalway.ie](mailto:sam.redfern@nuigalway.ie)

@psychicsoftware



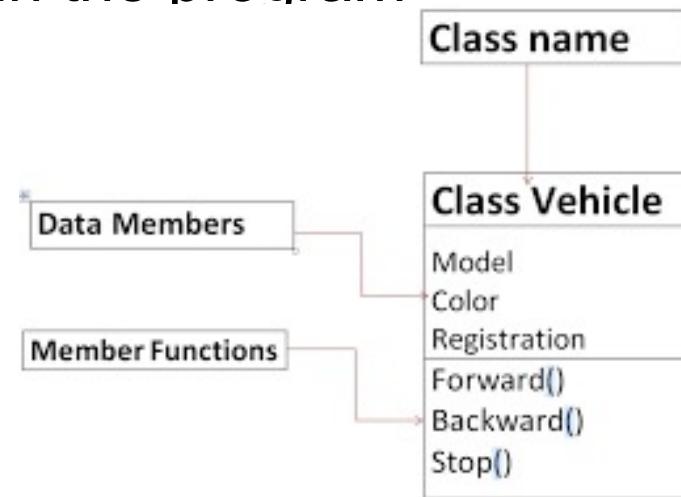
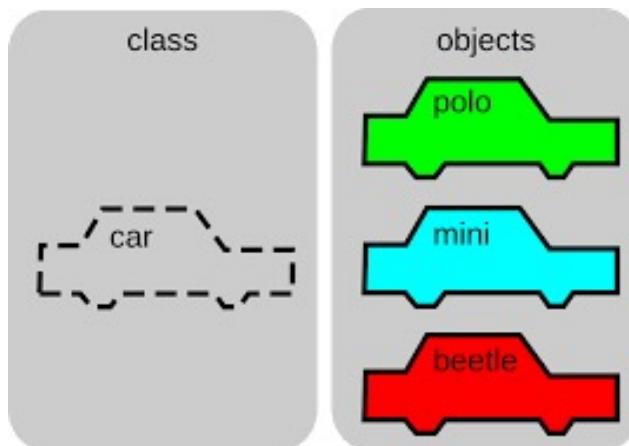
NUI Galway  
OÉ Gaillimh

# Last Week's Exercise

- Write a webpage which displays, alongside each other:
  - The time according to the user's browser
  - The time according to the server
- You should use Ajax to periodically retrieve the server's time (every 0.5 seconds) before updating the webpage with the two time values
  - Q: why not just do it every 1.0 seconds?
- Hint: recall the use of **setTimeout** to repeatedly (periodically) call a Javascript function

# OOP: Classes and Objects

- A **class** is like a blueprint for things in Object Oriented Programming (OOP)
- Classes have **properties**
- Classes have **methods**
- An **object** is an **instance** of a class, i.e. an actual live set of data about a thing in the program



# Classes in Javascript

- In Javascript, functions are objects
- So when you declare a function, you have an object!
- The keyword 'this' is used within an object's code to refer to its own instance

```
function myClass(var y) {  
    this.y = y; // .y is now a property of the object  
}
```

- This object also has an (implicitly created) class, which you can access through the prototype keyword

```
myClass.prototype.hello = function() {  
    // 'hello' is now a member function (i.e. a method) of the myClass class  
    alert("hello, my value of y is "+this.y);  
};
```

- To declare another instance of the myClass class, and use its hello method:

```
var obj = new myClass(6);  
obj.hello();
```

# Classless Objects

- You can also simply declare arbitrary data objects like this:

```
var o = {x:5, y:6, name:"Sam", data:[3,2,1], child:{a:2, b:6}};
```

- And of course you can access its members:

```
o.x = 6;  
alert(o.name);  
o.data[0] = 2;  
o.child.b++;
```

- And you can add more members at any time:

```
o.z = 20;  
o.child.c = [1,2,3];
```

# JSON and data organisation in Javascript

- JSON: “Javascript Object Notation”
- A syntax for specifying Javascript objects – now used in many languages
- This is JSON format:

```
var o = {x:5, y:6, name:"Sam", data:[3,2,1], child:{a:2, b:6}};
```
- **Serializing** is the act of turning a software object into a form that can be saved to file or sent across a network (i.e. as a series of bytes/characters).
- **De-serializing** is the inverse: turning a series of bytes/characters back into a software object
- Javascript provides functions for serializing/de-serializing objects as JSON:
- `JSON.stringify(o)`
  - returns a string containing a JSON representation of object o
  - E.g. for saving to disk or sending over the internet as text (e.g. using Ajax or Websockets)
- `JSON.parse(str)`
  - takes a JSON string and returns an instantiated Javascript object from it
  - E.g. for instantiating an object having read its JSON representation from disk or over the internet

# JSON Exercise

```
var car = { color:'red',  
wheels:[true,true,true,true],  
engine:{capacity:1000, power:200} };
```

Write a webpage which uses Javascript code to interact with this JSON object in order to:

- On a button click, use `alert()` to tell the user the color of the car object
- On a button click, use `prompt()` to let the user type a new color (as a string) and then change the car's color property to that value
- On a button click, tell the user whether the car's 4<sup>th</sup> and 5<sup>th</sup> wheels exist
- On a button click, tell the user the power of the car's engine
- On a button click, add a new property to the car, called 'value', and give it a number gathered from the user using `prompt()`
- On a button click, use `alert()` to display the entire car object as a serialized string

# Asynchronous Programming in Javascript

- Javascript was designed to deal with asynchronous situations
  - Waiting for a resource to arrive over the internet (recall: Ajax)
  - Waiting for the user to perform an action such as a button click (recall: onclick)
  - Waiting for time to pass (recall: setTimeout)
- So it has elegant ways to deal with these types of asynchronous situations, e.g. **callbacks**
- Callbacks are when we send a function as a parameter to another function, so that that other function can call the parameter function when required
  - Recall: `arr.sort(compare);`

# Callbacks using Named functions

```
<html>
<head>
<script>
    function start() {
        nextInSequence(1);
    }

    function nextInSequence(i) {
        document.getElementById("divOutput").innerHTML += i + "<BR>";
        if (i<12)
            setTimeout(nextInSequence, 500, i+1);
    }

</script>
</head>
<body onload="start()">
    <div id='divOutput'></div>
</body>
</html>
```

We're sending the `nextInSequence` function as a parameter to `setTimeout`

# Another Callbacks example (3 slides)

- Creating a customised (reusable) dialog box in HTML/Javascript

```
<div id="divAlert" style="position:absolute; width:100%; height:100%;  
overflow:hidden; left:0px; top:0px; text-align:center; display:none;  
background-color:rgba(0, 0, 0, 0.9); z-index:3;">  
    <div style="position:absolute; left:50%; top:50%; width:75%; transform:  
translate(-50%, -60%); z-index:4;">  
        <h1 id='alertHeading'></h1>  
        <p id='alertMsg'></p>  
        <a class="btn blue" onclick="onClickedAlertOk();">OK</a>  
        <a class="btn blue" onclick="onClickedAlertCancel();">Cancel</a>  
    </div>  
</div>
```

```
function alert(msg,title,okCallback) { // callbacks are optional
    document.getElementById("alertHeading").innerHTML = title;
    document.getElementById("alertMsg").innerHTML = msg;
    document.getElementById("btnAlertCancel").style.display = "none";
    document.getElementById("divAlert").style.display = "";
    confirm.okCallback = okCallback;
}

function confirm(msg,title,okCallback,cancelCallback) { // callbacks are optional
    document.getElementById("alertHeading").innerHTML = title;
    document.getElementById("alertMsg").innerHTML = msg;
    document.getElementById("btnAlertCancel").style.display = "";
    document.getElementById("divAlert").style.display = "";
    confirm.okCallback = okCallback;
    confirm.cancelCallback = cancelCallback;
}

function onClickedAlertOk() {
    document.getElementById("divAlert").style.display = "none";
    if (typeof confirm.okCallback!="undefined")
        confirm.okCallback();
}

function onClickedAlertCancel() {
    document.getElementById("divAlert").style.display = "none";
    if (typeof confirm.cancelCallback!="undefined")
        confirm.cancelCallback();
}
```

- Sending callbacks as named functions:

```
function clickedExitGame() {
    confirm("Are you sure?", "Exit Game", confirmedExitGame, cancelledExitGame);
}

function confirmedExitGame() {
    // here's where we actually exit.. somehow..
}

function cancelledExitGame() {
    // they changed their mind.. deal with it here..
}
```

- Or as anonymous functions:

```
function clickedExitGame() {
    confirm("Are you sure?", "Exit Game", function() {
        // here's where we actually exit.. somehow..
    }, function() {
        // they changed their mind.. deal with it here..
});
}
```

# Javascript ES6

- New ways to declare variables: **const** and **let**
- New Array methods: **map**, **reduce**, **filter**
  - Providing simpler syntax for manipulating arrays of data
- Alternatives to callbacks for asynchronous programming: **Promises**, **Async/Await**
  - We won't be looking at these, as they're a bit more complex than we have time for

# Surely ‘var’ is the only way to declare variables in Js?

- No, not anymore!
- A problem with ‘var’ is that its scoping rules are unusual
  - if declared inside a function, even within a loop etc., then a var will have ‘function scope’ (this is called ‘hoisting’, and some programmers dislike it)
  - this is at odds with most languages (e.g. Java) which have ‘block scope’
- The keyword ‘let’ can be used instead of ‘var’ if you want more normal (and thus, perhaps less error-prone) scoping
- Another new keyword for declaring variables is ‘const’
  - To create constants, which can never change after their initial declaration/assignment
  - Can remove errors caused by accidentally overwriting a variable
- See example on next slide

# Example

```
<html>
  <head>
    <script>
      function test() {
        var j = 0;
        let k = 0;
        const l = 20;
        for (var i=0;i<10;i++) {
          var j = i; // this will be the same variable as the previous j
          let k = i; // this will be a new, block-scoped variable
          l = i; // this line will cause a runtime error
        }
        alert("The value of j is: "+j);
        alert("The value of k is: "+k);
      }
    </script>
  </head>
  <body onload="test();">
  </body>
</html>
```

# .map(callback)

- A new method of Arrays
- For translating (mapping) all elements in an array to another set of values.
- Traverses the array from left to right invoking a callback function on each element. For each callback the value returned becomes the element in the new array.
- After all elements have been traversed, .map() returns the new array with all the translated elements

# .map() example

```
var fahrenheit = [0, 32, 45, 50, 75, 80, 99, 120];
var celsius = fahrenheit.map(
  function(elem) {
    return Math.round((elem - 32) * 5 / 9);
  }
);

// or in the new ES6 'arrow functions' syntax:
var celsius =
fahrenheit.map(elem => Math.round((elem - 32) * 5 / 9));

// content of celsius array is now:
// [-18, 0, 7, 10, 24, 27, 37, 49]
```

# .filter(callback)

- A new method of Arrays
- For removing specific elements of an array
- Traverses the array from left to right invoking a callback function on each element. The returned value must be a boolean identifying whether the element will be kept or discarded.
- After all elements have been traversed, .filter() returns a new array with all elements that returned true

# .filter() example

```
// this examples removes all duplicates in arr
var arr = [1,10,4,10,3,4,10,8,10];
var uniqueArray = arr.filter(
  function(elem, index, array) {
    return (array.indexOf(elem) == index);
  }
);
```

```
// or in new ES6 'arrow functions' syntax:
var uniqueArray =
arr.filter((elem,index,arr) => arr.indexOf(elem) == index);
```

# .reduce(callback, initialValue)

- A new method of Arrays
- For finding a cumulative or concatenated value based on elements across the array
- Traverses the array from left to right invoking a callback function on each element.
- The value returned is the cumulative value passed from each callback, added to initialValue.
- After all elements have been traversed, .reduce() returns the cumulative value

# .reduce() example

```
var rockets = [
    { country:'Russia', launches:32 },
    { country:'US', launches:23 },
    { country:'China', launches:16 },
    { country:'Europe(ESA)', launches:7 },
    { country:'India', launches:4 },
    { country:'Japan', launches:3 } ];

var sum = rockets.reduce(
    function(prevVal, elem) {
        return prevVal + elem.launches;
    }, 0
);

// In ES6 'arrow functions' syntax
var sum = rockets.reduce((prevVal, elem) => prevVal + elem.launches, 0);
```

# Some Revision Exercises..

# 1. DHTML (Dynamic HTML)

- Write a webpage which contains a text input field and a button
- When the button is clicked, use the window.alert method to tell the user what they typed into the text field

# 2. DHTML (Dynamic HTML) #2

- Write a webpage which contains a text input field and an empty `<div>` tag
- As the user types into the text input field (i.e. after each keypress), convert their text into a number
- If the conversion fails to produce a number, put an error message into the `<div>` tag
- If the conversion succeeds, show the "times table" of their number in the `<div>` tag, e.g:

You typed 4

1 x 4 = 4

2 x 4 = 8

3 x 4 = 12

# 3. Simple Animation

- Edit the provided webpage (next slide) in order to animate this goblin character by cycling through its 6 png files (please update 5 times per second)
- Now make the goblin appear to run across the screen, by moving the img as well as changing its src



# Base code for exercise on previous slide

```
<!DOCTYPE html>
<html>
  <head></head>
  <body>
    <div style='position:absolute; left:200px; top:200px;' id='theGoblin'>
      <img id='theGoblinImage' src='goblin_1.png'>
    </div>
  </body>
</html>
```

# 4. More simple animation

- Modify the provided webpage (next slide) in order to move the 'ball' image around within the bounds of the green <div> tag
- Initially, the ball should move, per second, 50 pixels to the right and 50 pixels downwards
- Whenever the ball hits an edge of the <div> tag, its movement in the appropriate direction should reverse.. i.e. hitting the right or left edge should reverse its horizontal movement, while hitting the top or bottom edge should reverse its vertical movement

# Base code for exercise on previous slide

```
<!DOCTYPE html>
<html>
  <head></head>
  <body>
    <div style='position:absolute; left:0px; top:0px;
width:600px; height:600px; background:#00FF00;'>
      <div style='position:absolute; left:200px; top:200px; '
id='theBall'>
        <img src='ball.png'>
      </div>
    </div>
  </body>
</html>
```

# 5. String Handling, Loops

- Using Javascript, take some text from the user, and tell them how many vowels are in that text (a,e,i,o,u,A,E,I,O,U)

# 6. Loops

- Make a webpage which obtains a number from the user
- The user should then be shown all prime numbers less than or equal to their number
- (A prime number is an integer that's greater than 1 that has no positive divisors other than 1 and itself).

# 7. Arrays, Loops

- Create an array containing 100 random integers, each between 1 and 999.
- In a <p>, display the numbers in sorted order